AD-753 115

SUMMARY OF THE SYSTEM, SUPPORT, AND SPECIAL-PURPOSE SOFTWARE USED BY THE SANGUINE SIMULATION FACILITY

Ira Richer, et al

Massachusetts Institute of Technology

Prepared for:

Electronic Systems Division
Department of the Navy

18 September 1972

Technical Note 1972-31

I. Richer
D. A. McNeill

Summary of the System, Support, and Special-Purpose Software Used by the Sanguine Simulation Facility

18 September 1972

Prepared for the Department of the Navy
under Electronic Systems Division Contract F19628-73-C-0002 by

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LEXINGTON, MASSACHUSETTS

DOCUMENT [illegible] DATA - [illegible]

([illegible] must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
Lincoln Laboratory, M.I.T.

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP
None

3. REPORT TITLE
Summary of the System, Support, and Special-Purpose Software Used by the Sanguine Simulation Facility

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Technical Note

5. AUTHOR(S) (Last name, first name, initial)
Richer, Ira and McNeill, Dale A.

6. REPORT DATE
18 September 1972

7a. TOTAL NO. OF PAGES
140

7b. NO. OF REFS
0

8a. CONTRACT OR GRANT NO. F19628-73-C-0002

b. PROJECT NO. 1508A

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
Technical Note 1972-31

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
ESD-TR-72-234

10. AVAILABILITY/LIMITATION NOTICES **Approved for public release; distribution unlimited.**

~~Distribution limited to U.S. Government agencies only; test and evaluation; 26 September 1972. Other requests for this document must be referred to ESD-DR-S.~~

11. SUPPLEMENTARY NOTES
None

12. SPONSORING MILITARY ACTIVITY
Department of the Navy

13. ABSTRACT

A computer facility, consisting of a Varian 620/i digital computer, associated peripherals, and extensive software, has been developed for analyzing and simulating communications systems. The software includes an operating system, general-purpose subroutines, and simulation programs. This report describes the software in detail, serving as a handbook for potential users and as a guide for setting up similar facilities.

14. KEY WORDS

Sanguine
Varian 620/i
ELF receiver
software

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

SUMMARY OF THE SYSTEM, SUPPORT, AND SPECIAL-PURPOSE SOFTWARE USED BY THE SANGUINE SIMULATION FACILITY

I. RICHER
D. A. McNEILL

Group 66

TECHNICAL NOTE 1972-31

18 SEPTEMBER 1972

[redacted]

LEXINGTON MASSACHUSETTS

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. The work was sponsored by the Department of the Navy under Air Force Contract F19628-73-C-0002.

ABSTRACT

A computer facility, consisting of a Varian 620/i digital computer, associated peripherals, and extensive software, has been developed for analyzing and simulating communication systems. The software includes an operating system, general-purpose subroutines, and simulation programs. This report describes the software in detail, serving as a handbook for potential users and as a guide for setting up similar facilities.

Accepted for the Air Force
Nicholas A. Orsini, Lt. Col., USAF
Chief, Lincoln Laboratory Project Office

TABLE OF CONTENTS

I. INTRODUCTION

In the process of developing and analyzing an ELF receiver, a simulation facility was developed consisting of a Varian 620/i computer, associated peripherals, and extensive software. The software was designed to permit rapid extraction, analysis, and display of pertinent data. Although our intent was to write software for the simulation project, the nature of the tasks encountered required that much general-purpose software be written. This software proves to be useful and adaptable to a wide variety of systems, both within and outside the field of communications. This report is a collection of memos describing the software in detail and is primarily written for those people who plan to use it or to set up a similar simulation facility.

Except for the assembler and some of the mathematics subroutines, all programs were written at MIT Lincoln Laboratory. The Varian assembler has been extensively modified for our purposes, and the mathematics subroutines were thoroughly checked and corrected if necessary. All the software is written in assembly language to minimize storage required by programs and processing time.

Section II lists the major hardware components in the simulation facility. Section III describes the computer operating system; update memos indicating changes and additions to the programs follow the original memos. Section IV details the general-purpose subroutines available, and Section V describes the specialized routines developed for the simulation program.

I. Richer wrote the DEBUG package, magnetic-tape routines, and the text editor. D. McNeill wrote the plotting package. Other major contributions are identified in the text. In addition, R. Teoste provided major contributions toward the framework of the simulation programs, and both he and C. Cappello wrote many of the simulation support routines.

II. HARDWARE

The software was written specifically for the hardware configuration at MIT Lincoln Laboratory; minor modifications may be necessary for operation on other configuration. Listed below are the major components of the simulation facility:

a) Varian 620/i digital computer with 32,768 (16-bit) words of memory, one accumulator, two index registers, hardware multiply and divide, direct-memory-access, "buffer interlace controller" (to permit block data transfer between memory and I/O devices), and 8 priority interrupts. Limitations on the Varian hardware permit block data transfers between a peripheral device and only the lower half of core; for that reason, DEBUG and the tape handling routines are resident in core from 31500_8-37777_8.

b) Teletype I/O terminal (model KSR 35 - 10 cps).

c) Computek CRT graphic display I/O terminal with a specially built 16-bit parallel interface. The interface was modified to accept a command so that hard copies can be obtained under program control.

d) Tektronix hard copy unit (for copies of CRT displays).

e) 4 PEC 9-track magnetic tape drives (25 ips, 800 bpi) operated with one controller. Modifications were made to enable the CPU to sense the tape drives online (see Section III-B).

f) Remex paper tape reader (300 cps).

g) Lincoln Laboratory built real-time clock (counts memory cycles for timing programs).

h) Hewlett-Packard D/A converter and X-Y plotter. This is an offline package for graphing data using paper tapes generated by the plotting subroutines.

Listings are obtained from high speed line printers at MIT Lincoln Laboratory's IBM 360 facility.

III. COMPUTER OPERATING SYSTEM

A. DEBUG Package and Magnetic-Tape Routines

1. Introduction

DEBUG, an "operating environment" for the Varian 620/i, greatly facilitates the loading and dumping of programs (or data) from magnetic tape, the examination and modification of the contents of registers*, and the testing and correcting of programs. Some particularly useful features of DEBUG are the ability to instruction-step through a program, the ability to interrupt a program in execution, and the ability to type out the contents of registers in a semi-symbolic format.

DEBUG is operated entirely from the teletype console. With DEBUG in operation, the user should consider the teletype to be his computer console: any operation that could be performed from the central-processor console (except for setting sense switches) can now be performed from the teletype. The purpose of DEBUG, in addition to providing many functions not available at the central-processor console, is to simplify the usual console operations.

There are two basic states of DEBUG: either a register is open or all registers are closed (i. e., no register is open). By an "open register" we mean the register currently being operated upon. Only one register can be open at a time. An instruction to DEBUG is either an argument followed by a command or just a command. The argument is an octal number (positive or negative) which represents the contents of a register or the address of a memory register. The command, always given by a single character, initiates the desired action, (or causes an error message to be typed). For example, the instruction

100,

means enter the argument 100 and prepare to accept the next argument.

* The term "register" is used to denote one of the core memory locations, one of the five "hardware" registers (A, B, X, program counter, and overflow indicator), or one of the special purpose registers provided by DEBUG (J, K, N, and T)

(Here the comma is the command; its function is explained in more detail below.) If all registers are closed, successive arguments are stored in an argument list. The argument list always contains five arguments, and as long as DEBUG is in control (i.e., so long as program execution or instruction-step are not initiated), the list remains intact unless it is specifically altered by the user. In other words, DEBUG uses the most recently entered value for each of the arguments in the list, and the user need enter only those arguments that are to be changed. For example, with all registers closed the four instructions

```
100,,,400,
```

would enter into the argument list 100 for Argl and 400 for Arg4 and would leave Arg2 and Arg3 intact.

The commands in DEBUG may be divided into three rough categories: those associated with register examination/change, those associated with program execution, and those associated with loading/dumping. In the next three sections this grouping will be used. In Section 5 the commands are summarized and listed in numerical order (according to their ASCII code). Section 6 provides details of the semi-symbolic output mode.

2. Register Examination/Change Commands

In order to examine or to change the contents of a register, the register must be open. As noted above, only one register may be open at a time. Some commands may be issued only when a register is open, some only when all registers are closed, and some in either situation.

In all the examples, underlined characters are those typed by DEBUG. In general, each line represents a distinct example; if, however, several lines constitute one example, then the lines are bracketed together (as in the second example under the /-command below). A carriage return and a line feed, where not obvious, are denoted by c.r. and l.f. respectively.

Command: P (for Program Counter)

Action:

Open and show the contents of the Program Counter (or P register); error if a register is already open.

Examples:

P/123	The P register contains 123_8 and is now open.
P/123 P?	Error if P is typed with a register open.

Commands: A (for A register)
B (for B register)
X (for X register)
V (for Overflow indicator)
J (for J address)
K (for K address)
N (for Number of registers)
T (for Trap address)

Action:

Similar to that for the P-command.

The P, A, B, X, and V "hardware" registers provided under DEBUG serve the same function as the corresponding registers that are provided at the actual computer console. The J, K, N, and T registers are special-purpose registers provided by DEBUG; their use is explained in Section 3 in connection with program execution and traps.

Command: / (slash)

Action:

If all registers are closed, open and show the contents of Argl; if a register is open, open and show the contents of the last-named register.

Examples:

1000/1234	Open and show register 1000.
X/1000 / 1000/1234	Open and show register X, then open and show the contents of X (viz. register 1000).

Command: , (comma)

Action:

Enter the last argument (if any), and prepare to accept the next argument or command. Commas are used to enter a sequence of arguments into the argument list or into successive registers.

Examples:

1000, 2000,	Arg1 is set to 1000, Arg2 is set to 2000, and Arg3 may be typed.
1000/1234 5001, 5322, ,	The value 5001 is stored in location 1000, 5322 is stored in 1001, location 1002 is left unchanged, and a value may be stored into location 1003 (i. e. register 1003 is now open).
1000/5001 1000, 2000/ 2000/5000 5001,	"Inserting a patch": ", JMP, 2000" is inserted at locations 1000 and 1001, and a patch program is started at location 2000.

Command: c. r. (carriage return)

Action:

Enter the last argument and close the open register (if any). DEBUG is now ready to accept Arg1. When a carriage return is typed, DEBUG also types out a line feed.

Examples:

1000/123 567, 1234 c. r. l. f.	567 is stored in register 1000, and 1234 is stored in register 1001 which is then closed.

1, 2, 3, 4, c. r. l. f.

The four arguments are entered in the argument list. The c. r. is used here in order to return to the start of the argument list (if, e. g., an error was made in Arg1).

Command: (space)

Action:

Enter the last argument.

Example:

1000/1234 555

If a space is typed after the 555, this value is stored in register 1000, but register 1000 remains open.

Command: - (minus sign)

Action:

Set the current argument negative.

Command: . (period)

Action:

Set the output mode to decimal, and if a register is open, type out the contents of that register. The output mode remains decimal until changed by an O or I command.

Examples:

10/100 . 64

Register 10 contains $100_8 = 64_{10}$.

10/123 100 . 64 1000 . 512

100_8 is stored in register 10 (by typing a space). This value is typed out in decimal, and it is changed to 1000_8, which is then also typed out in decimal.

11,2.2,33,	The three arguments 11, 22, and 33 are stored in the argument list, and the output mode is set to decimal. (Note that the period may be typed at any time and, since no register is open, has no effect other than to set the output mode.)

Command: O (letter "O" for "Octal")

Action:

Set the output mode to octal, and if a register is open, type out the contents of that register. The output mode remains octal until changed by an I or . command.

The value 177777_8 is always typed as -1 by DEBUG.

Example:

10/100 .64 ,-2 O177776	Register 10 contains $100_8 = 64_{10}$, and register 11 is set to $-2 = 177776_8$.

Command: I (for "Instruction")

Action:

Set the output mode to "instruction", and if a register is open, type out the contents of that register. The output mode remains "instruction" until changed by an O or a . command. The format and mnemonics used in this mode are discussed in detail in Section 6.

Examples:

100/50010 I STA 10	Register 100 contains the instruction ", STA, 10".
2000/54077 I STA 2100	Register 2000 contains the instruction ", STA, 2100" (addressing mode is relative to P).

Command: ; (semicolon)

Action:

If a register is open, enter the last argument and open and show the contents of the next register; error if no register is open.

Examples:

```
P/1000 100;
A/177776
```

P is set to 100 and A is opened. (The "hardware" registers are considered to lie in the following order: P, A, B, X, V.)

```
100/50010 I STA 10 ;
101/ JMP 1 ;
103/ LDA 11
```

(Note that since the JMP is a two-word instruction, the register following 101 is 103.)

Command: : (colon)

Action:

If a register is open, enter the last argument and open and show the contents of the previous register; error if no register open.

Example:

```
103/10011 I LDA 11 :
102/ HLT 1 :
101/ JMP 1 :
100/ STA 10
```

(The "previous" register is always one les than the open register. Thus register 102 (which contains 1), when entered from 103, is interpreted as a HLT instruction even though it is actually the second word of the JMP at location 101.)

Command: F (for "Find")

Action:

Search the registers from Arg1 to Arg2 for the value Arg3 masked by Arg4. That is, type all registers between Arg1 and Arg2 (inclusive) which agree with Arg3 in the bit positions marked by ones in Arg4. Thi command may be given only if all registers are closed. If Arg4 = -1 : 177777_8, the entire register must match Arg3. If Arg4 = 0, all regist match, and a number of consecutive registers may be examined with one instruction. Upon entry into DEBUG, and after a return from a trap (see T), from an instruction step (see S), or from an interrupt (see INT), the mask, Arg4, is set to 0. Typing (of output) may be suppressed by enabling Sense Switch 1.

Examples:

100, 103, , 0, IF 100/ STA 10 101/ JMP 1 103/ LDA 11 c. r. l. f.	Registers 100 - 103 are typed out in instruction mode.
0, 1000, 222, -1, F 77/222 777/222 c. r. l. f.	Registers 77 and 777 contain the value 222.
0, 1000, 1111, 7777, Fc. r. l. f.	No register from 0-1000 contains 1111 as the 12 least significant bits.

Command: W (for "Write")

Action:

Store Arg3 in all registers from Arg1 to Arg2 inclusive. This command may be given only if all registers are closed.

Example:

0, 100, -1, Wc. r. l. f.	$-1 = 177777_8$ is stored in 0-100.

Command: Any illegal character in DEBUG (e. g. "Z", "?", etc.)

Action:

Delete the argument being typed.

Examples:

123, 45, 66Z? 67	The argument 66 is discarded (DEBUG) responds by typing a question mark and a blank) and 67 is entered in its place.

1000/1234 43? ? ,4321	Register 1000 is left unchanged, and 4321 is stored in 1001.

Misc. Example:

10, 1000, -1, -1, F	Search registers 10-1000 for -1.
77/-1 c. r. l. f.	Register 77 contains -1.
100, Wc. r. l. f.	Write -1 into 100-1000. (Arg2 and Arg3 remain intact at 1000 and -1, respectively.)

3. Program Execution/Trap Commands and Program Interrupt

Described in this section are the commands used for stepping through a program and for continuing execution of a program. Also outlined are the use of the T register for setting a trap (instruction break) and the use of the J, K, and N registers for specifying the type-out after a trap. An example illustrating the use of these commands is given after the explanations. Finally, the program-interrupt feature is discussed.

Command: S (For "Step")

Action:

Starting from the current value of P, step through the number of instructions specified by Arg1; if Arg1 is not specified (or if it is specified as 0), step one instruction. All registers must be closed when this command is issued.

After the Step command is completed, DEBUG types out the new value of P, and returns control to the user (who may again perform any of the DEBUG operations). Note that since all input arguments are octal, the number of instruction-steps is the octal value of Arg1. For example, the command "10, S" will step through $10_8 = 8_{10}$ instructions.

In performing an instruction-step, DEBUG places a ", JMPM, BREAK" instruction immediately following the instruction to be executed. (The routine BREAK in DEBUG restores the locations that were overwritten by the JMPM instruction and then transfers control to the user after the specified number of steps have been executed.) Therefore, a program should never be stepped across an instruction that modifies either of the two addresses following the instruction, e.g. ", STA, *+2", "INR, *+1"; nor should a program be stepped across an instruction of the form ", JAP, *+3". Also, stepping across an instruction that does input from the teletype may give erroneous results.

Command: C (for "Continue")

Action:

Continue execution from the current value of P. If a trap has been set, execution stops the n-th time that the Trap is reached, where n = Argl if Argl is specified, and n = 1 otherwise. All registers must be closed when this command is issued

In order to set a trap at a particular instruction in a program, the address of the instruction is entered into the trap register T. To remove the trap, T is set to -1. When execution stops at a trap the contents of a number of registers are typed and control is returned to DEBUG. The user dictates, to a certain extent, the particular registers that are typed by DEBUG after a trap. The rules are as follows:

1) The value of P is always typed.
2) Depending upon the value contained in the "Number register" N, 0 to 4 of the remaining "hardware" registers A, B, X, V may be typed. Thus, if N = 0, none of these registers is typed; if N = 1, A is typed, if N = 2, A and B are typed, etc.
3) The contents of up to two memory registers may also be typed. This is especially convenient when the contents of certain locations must be inspected each time a trap is reached. The J and K registers hold the addresses of the memory registers to be typed,

with the value -1 signifying that no register is specified. (See the following example.)

Several warnings apply to the use of traps. Clearly a trap should not be set in the middle of a two-word instruction, nor should one be set at an address if during execution the program jumps (branches) to the trap address +1. In addition, if execution begins from the trap address or from the trap address +1, then the program is essentially instruction-stepped past trap address +1 (and then a ", JMPM, BREAK" is placed at the trap address). Therefore, the same cautions given for the Step command apply for traps if execution is to continue from the trap address. Finally, if a trap was set and if program execution terminates without a return to DEBUG, then the locations (trap address) and (trap address +1) must be returned to their original values in order to restore the program.

Example:

Assume that the following program is stored in memory:

Location	Contents	Symbolic Code		
100	005001		TZA	
101	050112		STA	COUNT
102	040112		INR	COUNT
103	001001		JOF	*+4
104	000107			
105	001000		JMP	*-3
106	000102			
107	005000		NOP	
110	005000		NOP	
111	000000		HLT	
112		COUNT	BSS	1

The following operations illustrate the use of the commands associated with program execution.

```
P/1000 100 c. r. l. f.      Set P register to entry point.
V/0 c. r. l. f.             Make sure V = 0.
3, Sc. r. l. f.             Step 3 instructions.
P/103c. r. l. f.
112/1 c. r. l.f.            Check that COUNT = 1.
T/-1 102c. r. l. f.         Set a trap at location 102.
J/-1 112;                   Set J to the address of COUNT.
K/-1 c. r. l. f.            Leave K with no address specified.
N/4 0 c. r. l. f.           At a trap, type no registers other than
                            P (and 112).
100, C                      Continue execution and break after the
112/100                     trap is reached 100₈ times. As expected,
P/102 c. r. l. f.           COUNT = 100₈.
T/102 107c. r. l. f.        Move the trap to location 107.
N/0 4c. r. l. f.            Show all registers at next trap.
C                           Continue execution.
112/100000                  Location 107 is reached with COUNT =
P/107                       77777 + 1. (The JOF instruction resets
A/0                         the overflow indicator V.)
B/0
X/-1
V/0c. r. l. f.
T/107 -1 c. r. l. f.        Remove the Trap.
```

Command: INT (special interrupt key)

Action:

Interrupt the program in execution, type the contents of P, A, B, X, and V, and return control to DEBUG. The interrupt key is disabled when DEBUG is in control (except during loading and dumping); the key is enabled when program execution begins.

The primary use of the program-interrupt feature is to temporarily halt the execution of a program that is suspected (or known) to contain an error. For example, if a program seems to be looping indefinitely, execution could be interrupted and the program could be instruction-stepped until the error was located. If possible, the error could then be corrected and execution could be re-initiated from the start or from any intermediate point.

4. Load/Dump and Miscellaneous Commands

This group of commands allows for loading and dumping of programs (or data) from magnetic tape, for rewinding a tape, and for re-starting DEBUG. These commands may only be given with all registers closed.

Command: L (for "Load")

Action:

Load the program stored on tape unit = Arg1, file number = Arg2. DEBUG will respond with one of three messages:

1) If loading is successful, the entry point is stored in P and this value is typed out. The C-command may then be used to initiate execution.
2) If Arg1 represents an invalid tape unit (only 0 and 1 are valid), or if there is a read error, the error message "TAPE" will be issued.
3) If the specified file is not a binary program file (viz. the binary output from the assembler or the output from a dump in DEBUG), the message "FILE" will be typed.

In order to avoid possible errors, the trap register is reset to -1 when the L-command is given.

Example:

0, 0,L	Load from tape 0, file 0.
FILE c. r. l. f.	This file is not a program.
0, 1,L	Load from tape 0, file 1.
P/1353 c. r. l. f.	Load is complete; entry point is 1353.

Command: D (for "Dump")

Action:

Dump the block of registers from Arg1 to Arg2 inclusive, with entry point set to Arg3; the output tape and file are specified in Arg4 and Arg5, respectively. Only the last block in a binary file contains the entry point, so Arg3 should be set to -1 for all but the last block. Thus, if a program comprises several non-contiguous areas of core, several D-commands must be given. After the last block is dumped (with Arg3 = entry point ≥ 0) DEBUG types a carriage-return/line-feed; after intermediate blocks, DEBUG types "--" to indicate that the dump file is incomplete.

If Arg4 represents an invalid tape unit, or if there is a write error, the message "TAPE" is typed.

Example:

The following sequence will dump registers 100-200, 300-400, and 1000-2000 onto tape 1, file 2. The entry point is 111:

100, 200, -1, 1, 2,D-- 300, 400, D-- 1000, 2000, 111, D c. r. l. f.

Note that it is not necessary to re-enter Arg4 or Arg5 after the first command. (Also, Arg3 need be entered only for the first and last blocks.)

Command: R (for "Rewind")

Action:

Rewind the tape unit specified by Arg1. (If Arg1 represents an invalid unit, "TAPE" is typed.)

Rewinding should always be performed by DEBUG rather than manually because certain parameters that define the tape position must be reset.

Command: Q (for "Quit")

Action:

Restart DEBUG. (The tapes are rewound, all buffer areas are cleared, and DEBUG itself is initialized.)

5. Summary of DEBUG Commands

Command	Meaning. [Page reference]
c. r. (carriage return)	Enter the last argument, close the open register (if any), and prepare to accept Argl. [4]
(space)	Enter the last argument. [5]
,	Enter the last argument and prepare to accept the next argument or command. [4]
-	Argument is negative. [5]
.	Set output mode to decimal. [5]
/	Show the contents of. [3]
(0-7	Octal digits for input arguments.)
:	Enter the last argument and open and show previous register. [7]
;	Enter the last argument and open and show next register. [7]
A	A register. [3, 7]
B	B register. [3, 7]
C	Continue execution. [10]
D	Dump a block of core. [14]
F	Find a masked value. [7]
I	Set output mode to "instruction". [6, 17]
J	J address (register to be typed after a trap). [3, 10]
K	K address (register to be typed after a trap). [3, 10]
L	Load from tape. [13]
N	Number of registers to be typed after a trap. [3, 10]
O	Set output mode to octal. [6]
P	Program counter. [3, 7]
Q	Restart DEBUG. [15]
R	Rewind a tape. [14]
S	Instruction step. [9]
T	Trap address. [3, 10]
V	Overflow indicator. [3, 7]
W	Write a value into memory. [8]
X	X register. [3, 7]
INT (special switch)	Program interrupt. [12]
Any other character	Delete the last argument. [8]

6. Format and Mnemonics for "Instruction" Output Mode

The opcode mnemonics used in the "instruction" output mode are those recognized by the assembler (with the four altered long-shift opcodes). A double question mark (? ?) is used if the opcode is illegal. An asterisk (*) following the opcode indicates that the address in the variable field is indirect. Addresses and data in the variable field are always typed out in octal. Thus, ", LSRA, 11" means Logical-Shift-Right-A $11_8 = 9_{10}$ places.

Register-change instructions are typed in the following format:

OPCODE SOURCE, DESTINATION

Opcode is either ZERO, MERG, INCR, COMP, or DECR, and Source and Destination are the source registers and destination registers. For the ZERO opcode, only destination registers are specified. Finally, if the instruction is conditi^nal on the setting of the overflow indicator, "OF" is typed preceding the source registers.

Examples:

ZERO	XA	Zero in X and A
DECR	, A	-1 in A
MERG	X, B	Transfer X to B (TXB)
INCR	X,XA	Increment X and bring into A
INCR	OF, A, A	Increment A if overflow set (AOFA)

Two forms are used for the "conditional" instructions jump, jump and mark, and execute. If no conditions are specified, then the mnemonic JMP (or JMPM or XEC) is used; if conditions are specified, then the mnemonic JIF (or JIFM or XIF) is used, and the conditions are typed in the variable field preceding the address.

Mnemonic	Condition
OF	Overflow
AP	$A \geq 0$
AN	$A < 0$
AZ	$A = 0$

Mnemonic	Condition
BZ	B = 0
XZ	X = 0
S1	Sense Switch 1
S2	Sense Switch 2
S3	Sense Switch 3

Examples:

JMP	12345	Jump unconditionally.
JIFM	AZ, 123	Jump and mark if A = 0.
XIF	S1, AN, BZ, 456	Execute 456 if Sense Switch 1 and A < 0 and B = 0.

MAGNETIC-TAPE ROUTINES

Two sets of magnetic-tape routines have been written for the Varian 620/i. The file-handling routines provide a very convenient means for storing and retrieving data from tape. In using these routines the programmer need not concern himself with the details of buffer allocation, tape positioning, and error checking. The tape-handling routines are lower-level programs that permit more flexible use of the tapes but that require much more bookkeeping by the programmer.

Given below is the necessary programming information for the two sets of routines. Information and specifications on the magnetic tape units and on the method and format of data storage on tape can be found in Varian's "Magnetic Tape Controller" manual.

1. File-Handling Routines

Presented in this section are the specifications and the calling procedures for the following routines and system locations (all symbols are recognized by the assembler):

$RDIR	read directory	Directory routines
$WDIR	writc directory	
$OPEN	open file	Data-handling routines
$DATA	data to or from file	
$CLOS	close file	
$LOAD	load from tape	Dynamic loader
$INIT	initialize system	Initialization routines
$RESET	rewind a tape	
$DOUT	eliminate DEBUG	Routines for elimination and restoration of part of system
$SIN	reload system	
$RETN	used in error recovery	Memory registers holding system addresses
$SYS	starting address of system	
$DEBUG	entry point for DEBUG	

Preceding the specifications are some introductory remarks and explanations of the directory routines $RDIR and $WDIR and of the data-handling routines $OPEN, $DATA, and $CLOS. The use of the remaining routines and of the system locations is straightforward enough that they do not require explanations beyond those given in the specifications.

Information on magnetic tape is arranged in files, with each file comprising one or more records. For example, a tape file could be the text for a source-language program, a binary (assembled) program, or the output data produced by a program. The user must keep track of the files on his tapes. That is, he must maintain an up-to-date list of the contents of each active file. Since the file-handling routines perform all the necessary buffering chores, the programmer need not be concerned with the nature or number of records in his files.

In general it is not possible to recover space on a tape. That is, if the information written on a particular file is no longer needed, the space occupied by this file cannot be used for a new file even if the old and new files are of the same length. The basic reason is that because of the relative inaccuracies in positioning a tape, some of the useful data written beyond the old file may be destroyed when the old file is overwritten. (Of course, the space occupied by the last tape file may be recovered since there is nothing of interest beyond that point.) It is frequently desirable to have a file that can be updated dynamically (i. e. under program control) but that occupies a fixed position on the tape. This file could, for example, contain a list of the active files on the tape. The routines $RDIR and $WDIR permit the reading and writing of such a directory on tape. These directory routines may be used only with tapes that have been properly formatted.[*] The directory occupies one record (record 3, file 0) and is read or written in one unbuffered operation. The first word of the directory is the count - i. e., the number of words that follow. For example, a five-word

[*] Formatted tapes containing the system programs and a blank directory will be provided to users of the Varian 620/i.

directory would occupy six memory registers, with the first register containing the value 5. A typical program might read in the directory, refer to the directory and perhaps modify its contents (and also its count), and then write it out. If the directory is not modified, it need not be re-written.

The three routines that will be used most often - $OPEN, $DATA, and $CLOS - provide for data transfer to and from tape. $OPEN is used to "open" a tape file - i.e., to allocate the necessary buffer space for forthcoming I/O operations. Up to eight files may be open simultaneously. Certain information must be given to $OPEN in order to specify the file to be opened: the tape unit, the file number, whether the file is to be a read file (input from tape) or a write file (output to tape), and, if a write file, the type of file (binary program, text, etc.). This information is passed to $OPEN in registers. Upon return, $OPEN passes an identification word (ID) for the file. After a file has been opened, it is referred to by its ID. Thus, to store or retrieve the next item of data, $DATA is called with ID as the only calling parameter. When operations on a file are complete, $CLOS is called in order to free the buffer space (and to write an end-of-file mark for a write file).

With these routines, as with any routines that perform tape I/O, provision is made in case tape errors occur. If the tape operations are completed without error, then the routine returns normally - to the instruction following the call. However, if more than a fixed number of tape errors occur,* the routine returns to the error-return address that is specified with the call. An error return can be either non-fatal or fatal. Recovery from a non-fatal error is possible, provided that the program can tolerate the loss of one record of data; recovery from a fatal error is not possible. The possible causes of an error return are as follows (errors are fatal unless otherwise noted):

- Tape-positioning error (desired position cannot be found).
- Read error (i.e., position is correct, but a read error occurred). This error is non-fatal.

* At present, up to four tape errors are permitted.

- Tape-unit failure.
- No buffers available (during a call to $OPEN).

(An error return from a write-file operation is always fatal.) Note again that an error return results not from a single tape error, but from some combination of the above errors.

In order to recover from a non-fatal error, the program should execute the instruction ", JMP, $RETN". ($RETN is a special register which contains the proper address for a re-try of the I/O operation.) If recovery is not desired, then the file must be closed by the program. After a fatal error - and in particular after an error in $CLOS itself - the file is always closed prior to the return. In this case an additional call to $CLOS will not cause any errors.

Outlined below is a typical instruction sequence that illustrates the use of the data-handling routines and the error recovery procedure.

```
PROG      CALL     $INIT            Initialize
           .
           .
           .
          CALL     $OPEN, ERROR     Open file
           .
           .
           .
NEXT      CALL     $DATA, ERROR     Next data item
          JOF      DONE             End-of-file
           .
           .                        Process data
           .
          JMP      NEXT
ERROR     JAN      FATAL            Recovery impossible
           .                        Issue message to show that
           .                          data has been lost
           .
          JMP      $RETN            Try to recover
```

```
FATAL       .
            .                                  Issue appropriate message
            .
DONE        CALL       $CLOS, ERROR            Close file
            .
            .
            .
            JMP        $DEBUG                  Return to DEBUG
            END        PROG
```

In this example, if recovery was not desired, then all error returns would be to FATAL.

Specifications and Parameters for File-Handling Routines

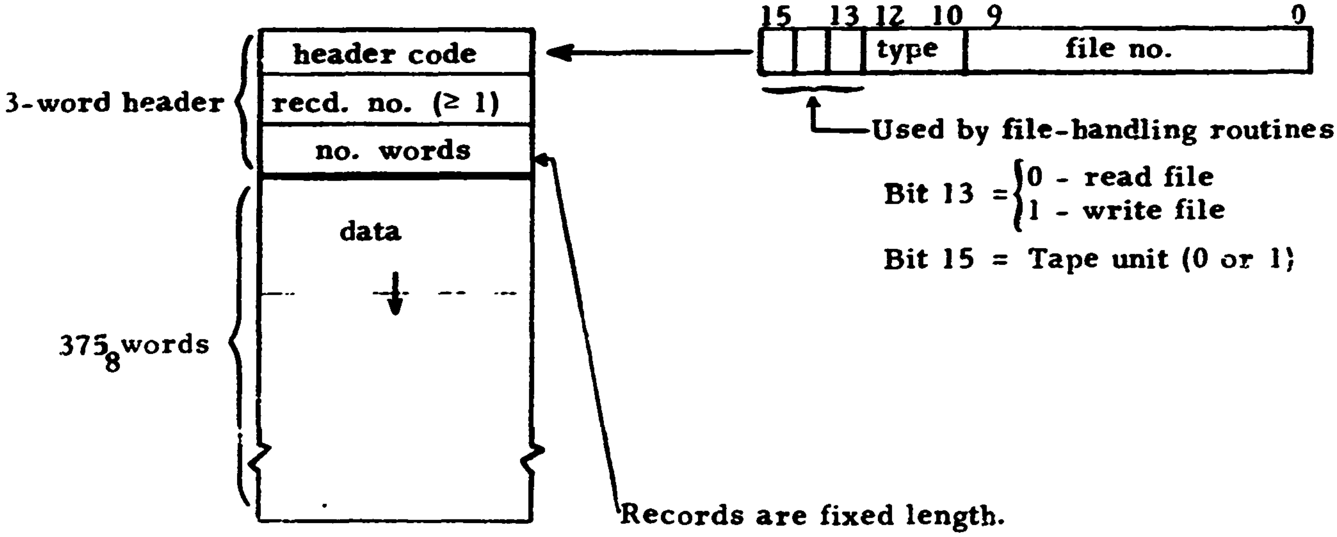
Tape unit·	**0 or 1**
File number:	**0 - 1777_8 inclusive**
File type:	**0 -binary program** **1 -text** **2 - 7 -unassigned**
File ID:	**>0 (returned by $OPEN; used in calls to $DATA and $CLOS)**
Buffer size:	**$400_8 = 256_{10}$ words**
Buffer allocation:	**Backwards from ($SYS), highest locations first. (See Core Map and description of $SYS.)**
Maximum no. buffers:	**8**
Allowed directory size:	**1-40000_8 words (approximate), including count. First word of directory is the count - i. e. the number of words following. A blank directory contains 0 as its first and only word.**
Number of tape errors for an error return:	**4**

Tape Format and Record Format

Formatted tape

File 0, record 1:	System programs (DEBUG, File-Handling Routines, Tape-Handling Routines).
File 0, record 2:	Interrupt instructions and tape-routine addresses.
File 0, record 3:	Directory.
Files 1 and up:	Available for use.

Format of records written by system



Used by file-handling routines

Bit 13 = {0 - read file, 1 - write file}

Bit 15 = Tape unit (0 or 1)

Records are fixed length. This is the number of actual data words in the record.

$\left\{\begin{array}{l}\underline{\text{\$RDIR}}\\ \underline{\text{\$WDIR}}\end{array}\right\}$ $\left\{\begin{array}{l}\text{Read}\\ \text{Write}\end{array}\right\}$ **directory from tape.**

Calling sequence: CALL $\left\{\begin{array}{l}\text{\$RDIR}\\ \text{\$WDIR}\end{array}\right\}$, (ERROR RETURN)

Entry parameters:
(B) = Starting location of directory.
(X) = Tape unit.

Return parameters: None. (Error return is always fatal.)

Registers used: A, OF

Remarks: The starting location of the directory contains the directory count - i. e. the number of words in the directory (excluding the count itself). Thus, count = 0 signifies an empty directory, and in general

final directory loc. = starting loc. + count.

These routines may only be used with formatted tape.

$OPEN — Open a file on magnetic tape and allocate the necessary buffer space.

Calling sequence: CALL $OPEN, (ERROR RETURN)

Entry parameters:

(A) = File type (≥ 0) if a write file;
= -1 if a read file.
(B) = File number.
(X) = Tape unit.

Return parameters:

Normal return -

(A) = File type.
(B) = Header code (see Record Format information).
(X) = File ID $\neq$ 0.

Error return -

(A) ≥ 0 if error is non-fatal;
= -1 if error is fatal (file is closed).
(B) = Header code.
(X) = ID word (unless no buffers available).
(OF) = Set if no buffers available. (This is a fatal error.)

Registers used: A, B, X, OF

$DATA

Store or retrieve data from a file.

Calling sequence: CALL $DATA, (ERROR RETURN)

Entry parameters:

(A) = Data (if a write file).
(X) = File ID.

Return parameters:

Normal return -

(A) = Data (new data if a read file, original data if a write file).
(OF) = Set if end-of-file on a read file;
= Reset otherwise.

Error return -

(A) ≥ 0 if error is non-fatal;
= -1 if error is fatal (file is closed).

Registers used: A, OF

$CLOSE

Close a file (or close all files) and free the appropriate buffer area.

Calling sequence: CALL $CLOS, (ERROR RETURN)

Entry parameters: (X) = File ID if a single file is to be closed;
= 0 if all open files are to be closed.

Return parameters:

Normal return - None.

Error return - (A) = -1 (all errors fatal).
(X) = ID of file that caused error.
(This file has been closed.)

Registers used: OF (A and X may be altered if an error return occurs.)

Remarks: An error return can only occur for a write file. If all files are to be closed and an error return occurs, X contains the ID of the first file that caused an error; an additional call to $CLOS, again with (X) = 0, must then be made in order to close the remaining open files.

$LOAD Dynamic loader from magnetic tape.

Calling sequence: CALL $LOAD, (ERROR RETURN)

Entry parameters: (B) = File number.
(X) = Tape unit.

Return parameters:
Normal return - (A) = Entry point.
Error return - (A) = 0, OF reset if tape error;
= 0, OF set if no buffers left;
= -1 if wrong file type (or if file is not complete).

Registers used: A, OF

Remarks: $LOAD requires one buffer during execution.

<u>$INIT</u> Initialize the tape system.

Calling sequence: CALL $INIT

Entry parameters: None.

Return parameters: None.

Registers used: None.

Remarks: $INIT rewinds the tapes, clears the buffers, and initializes certain system parameters. This routine should be called at the beginning of any program that uses the tapes if the status of the tapes or of the buffers is not known.

$RESET	Rewind a tape.
Calling sequence:	CALL $RESET
Entry parameters:	(X) = Tape unit.
Return parameters:	(A) = 0.
Registers used:	A
Remarks:	In addition to rewinding a tape, $RESET resets certain system parameters. When the file-handling routines are being used, tape rewinds should be performed only by $RESET or by $INIT.

$DOUT	Logically eliminate DEBUG from core.
Calling sequence:	CALL $DOUT
Entry parameters:	None.
Return parameters:	None.
Registers used:	A
Remarks:	This routine is called if the space occupied by DEBUG is required during execution of a program. (With DEBUG eliminated, the program-interrupt feature is of course disabled.) If $DOUT is called, then as its final instruction, the program in execution should reload DEBUG by calling $SIN.

$SIN — Reload the system and transfer control to DEBUG.

Calling sequence: JMP $SIN

Entry parameters: (X) = Tape unit.

Remarks: $SIN is called as the final instruction of a program that eliminated DEBUG. Note that this routine is called with a JMP instruction. If no tape errors occur, $SIN transfers control to DEBUG. If a tape error does occur, $SIN halts at the start of the bootstrap loader.

$RETN — Location used in recovery from a non-fatal error return.

Usage: JMP $RETN

Remarks: Register X (which contains the file ID when the jump to the error return is made) must contain the file ID. $RETN may be referenced only after a non-fatal error.

$SYS — Register containing the starting address of the system.

Typical usage: LDAE $SYS

Remarks: A program in execution may use all memory registers up to, but not including, the address held in $SYS. The program must allow space for the greatest number of buffers that will be used at any one time (one buffer for each open file plus one additional buffer if $LOAD is called). Buffers are allocated "backwards" from $SYS, highest locations first. (See Core Map.) A sample instruction sequence is

```
        .
        .
        .
LDAE    $SYS      1st system address
SUBI    N*0400    At most N files will
DAR                  be open simultaneously
STA     LAST      Last usable address
        .
        .
        .
```

The location LAST now holds the address of the last (highest-numbered) memory register that may be used by the program.

$DEBUG Entry point for DEBUG.

Usage: JMP $DEBUG

Remarks: After completing execution, all programs should return to DEBUG, either by a jump to $DEBUG or, if DEBUG has been eliminated by $DOUT, by a call to $SIN.

2. Tape-Handling Routines

In this section the specifications and calling procedures for the following basic tape routines are given:

$RECD $FILE $REW	- Tape positioning routines
$READ $WRITE $FMRK	- Read/Write routines
$ECHK	- Error checking routine
$UNIT	- Sense tape unit ready
$DEV $UDEV	- Special interpretive routines

These routines give the programmer essentially complete control over tape operations. The read/write routines and the positioning routines (except for $FILE) only initiate the requested action. If necessary the program should check for completed action and for possible errors. In the specifications below, the timing, if applicable, is given as the minimum number of cycles required for the routine to initiate the appropriate action and then return. (Execution will proceed in the minimum time if the referenced tape unit is in the ready state and if, for a read or write operation, the BIC is also in the ready state.) As with the file-handling routines, the routine-names are recognized by the assembler. It should be noted that if the file-handling routines are not used, then the space they occupy may be utilized. If this is done, then the contents of $SYS are no longer meaningful (see the Core Map to obtain the address of the last usable memory register), and the system should be reloaded (by jumping to $SIN)[*] after execution is completed.

[*] The routine $SIN is located after the tape-handling routines, i. e., immediately preceding the bootstrap loader.

$RECD Move tape a specified number of records.

Calling sequence: CALL $RECD

Entry parameters:

(A) = Number of records:

- (A) > 0 - move forward (A) records;
- (A) = 0 - no movement;
- (A) < 0 - move backward -(A) records.

(X) = Tape unit.

Return parameters:

(A) = 0 if move completed;

< 0 if beginning-of-tape encountered (during a backward move); -(A) = number of records left to move.

Registers used: A

Remarks:

If the end-of-tape mark is reached, the tape is rewound and forward motion is resumed. That is, the forward move "wraps around" from the end to the beginning of the tape.

When $RECD returns, the tape unit is performing the motion associated with the last record of the requested move.

$FILE

Move tape forward a specified number of files or position tape at a specified file.

Calling sequence: CALL $FILE

Entry parameters:

(A) = Number of files:

- (A) >0 - move forward (A) files;
- (A) =0 - no movement;
- (A) <0 - position tape at file number = complement(A).

(X) = Tape unit.

Return parameters: (A) = 0.

Registers used: A

Remarks: No provision is made for a backward move because the hardware associated with the tape units cannot sense a file mark during backward tape motion. If $FILE is called with (A) <0, the tape is rewound and then advanced complement (A) files. As with $RECD, the tape is considered to wrap around from end to beginning.

$REW	Rewind a tape.
Calling sequence:	CALL $REW
Entry parameters:	(X) = Tape unit.
Return parameters:	None.
Registers used:	None.
Remarks:	When $REW returns, either the tape unit is ready at the load point or it is in the process of rewinding.

$READ **$WRITE**	**Read** **Write** **a record.**
Calling sequence:	**CALL** **$READ** **$WRITE**
Entry parameters:	**(A) = Final memory address.** **(B) = Starting memory address.** **(X) = Tape unit.**
Return parameters:	**None.**
Registers used:	**None.**
Timing:	**$READ - 50. 75 cycles (minimum).** **$WRITE - 57. 75 cycles (minimum).**
Remarks:	**The contents of locations (B) to (A) inclusive are filled ($READ) with data from the next tape record, or they are written ($WRITE) as the next record on the tape. To read a record of unknown length, set (A) large [e. g. (A) = ($SYS)-1] in the call to $READ, and then call $ECHK to obtain the actual final address.**

$FMRK	**Write a file mark.**
Calling sequence:	**CALL $FMRK**
Entry parameters:	**(X) = Tape unit.**
Return parameters:	**None.**
Registers used:	**None.**
Timing:	**49. 25 cycles (minimum).**

$ECHK

Check for errors (after $READ or $WRITE).

Calling sequence: CALL $ECHK, (ERROR RETURN)

Entry parameters: (X) = Tape unit.

Return parameters:

Normal return - (A) = -1 if operation was completed normally;
= Final memory location (≥ 0) if operation terminated prematurely.

Error return - (A) = 0.

Registers used: A

Timing:

Normal return - 60.75 cycles (minimum).
Error return - 47.5 cycles (minimum).

Remarks: After a normal return, the caller must check whether $A < 0$ or $A \geq 0$ and must then take appropriate action. If $(A) \geq 0$, then A contains the final memory location that was used in the previous I/O operation - for example the last location filled in $READ. After a read operation an error return signifies a tape read error, and after a write operation it signifies that the write-enable ring is not present.

$UNIT	**Sense tape unit ready.**
Calling sequence:	CALL $UNIT
Entry parameters:	**(X) = Tape unit.**
Return parameters:	**None.**
Registers used:	**None.**
Timing:	12.25 cycles (minimum).
Remarks:	**$UNIT does not return until the specified tape unit is in the ready state.**

$DEV

Store a device number into an instruction.

Calling sequence:

```
CALL    $DEV,INSTRUCTION
```

or, equivalently,

```
CALL    $DEV
INSTR   . . .
```

Entry parameters: (X) = Device number: $0 \le (X) \le 7$.

Return parameters: None (INSTRUCTION is altered).

Registers used: None (See Remarks).

Timing: 20 cycles.

Remarks: $DEV is an interpretive routine that simplifies programming if two or more of the same type device are part of the computer system. Bits 0 - 2 (the device number) of INSTRUCTION are masked out and replaced by bits 0 - 2 of register X. The modified INSTRUCTION is set back at the original location, and $DEV returns by jumping to this location. ($DEV masks out bits 3 - 15 of register X, and hence upon return these bits are 0.) For example, the following sequence tests for end-of-tape on the unit number held in location TAPE:

```
    .
    .
    .
 LDX      TAPE        Tape unit in X.
 CALL     $DEV        Set unit no. in SEN instr.
 SEN      0510,EOT    Jump to EOT if end-of-tape.
    .
    .
    .
```

$UDEV	**Wait for tape unit to enter ready state and then proceed as for $DEV.**
Calling sequence:	CALL $UDEV, INSTRUCTION
Entry parameters:	(X) = Tape unit = Device number.
Return parameters:	None (INSTRUCTION is altered).
Registers used:	None (See Remarks under $DEV).
Timing:	40. 25 cycles (minimum).

COLD-START PROCEDURE

1. Turn computer power on, enable memory, and press STEP and then SYSTEM RESET.

2. Turn power on for the teletype and magnetic-tape units.

3. Mount a formatted tape on unit 0 (or on unit 1 if the indicated changes are made in the bootstrap loader), mount any tape on the other unit, and bring each tape to its load point.

4. Enter the bootstrap loader:
   (a) Enable REPEAT.
   (b) Set the U register to 54000 (STA relative to P).
   (c) Set the P register to* Y7767.
   (d) Enter a bootstrap instruction (see next page) into the A register.
   (e) Press STEP to enter the instruction in memory.
   (f) Repeat steps (d) and (e) until all bootstrap instructions are entered.

5. Set* (P) = Y7770, (U) = 0, (A) = Y7777, (B) = Y4000, and (X) = 0.

6. Press RUN. If the system loads properly, DEBUG will begin execution. If there is a tape error, execution will halt with (P) = Y27770. In this case, rewind the tape, set A and B as in Step 5, and press SYSTEM RESET and RUN in order to re-attempt the load.

* Y = 2 for 12K memory, Y = 3 for 16K memory, etc.

BOOTSTRAP LOADER

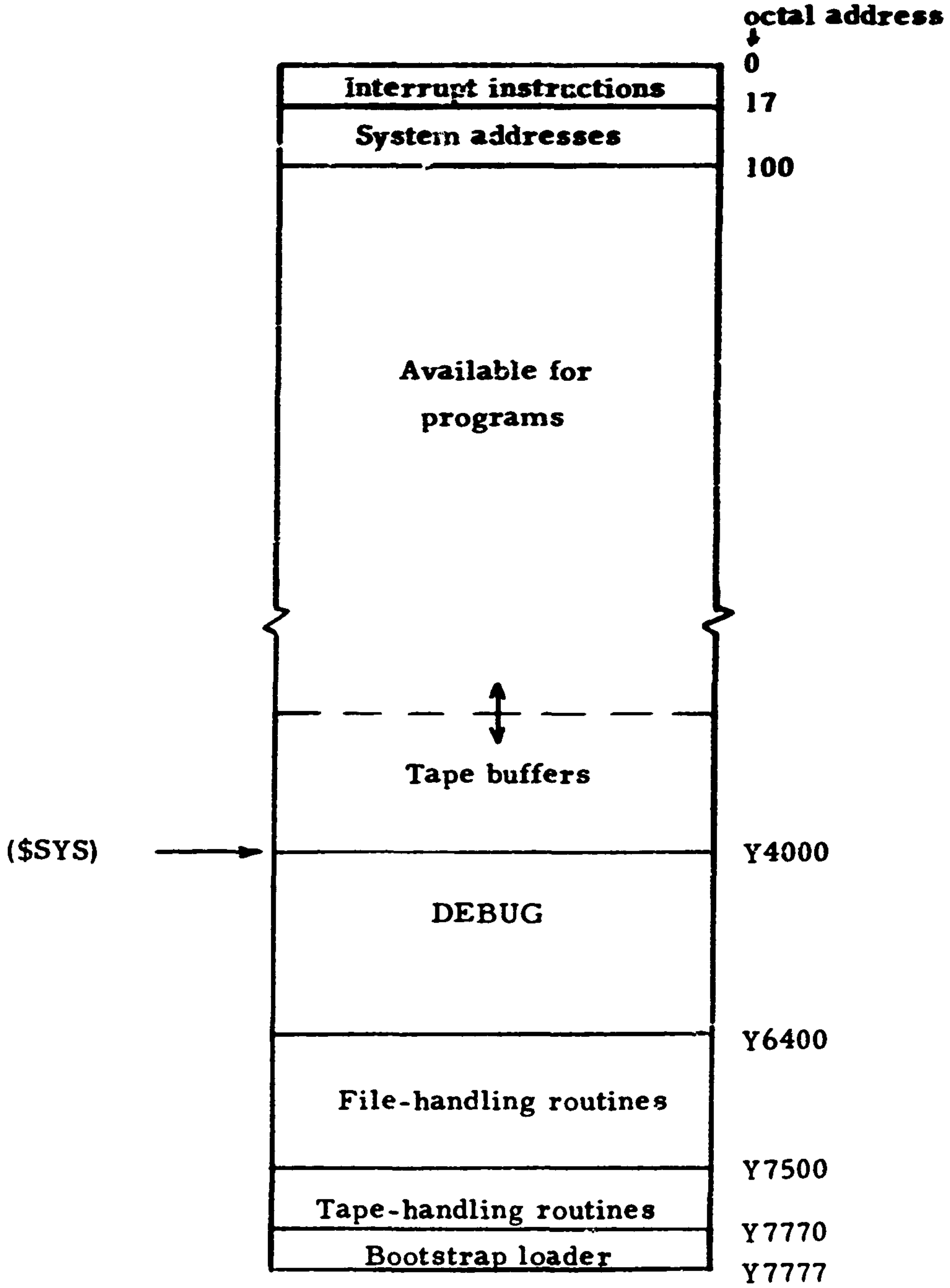
Location	Contents	Symbolic Code		
Y7767	000000	HLT		
Y7770	103220	OBR	020	Starting BIC location
Y7771	103121	OAR	021	Final BIC location
Y7772	100020	EXC	020	Enable BIC
Y7773	100010	EXC	010	Read a record
Y7774	101210	SEN	0210, $SIN	When ready, call $SIN to
Y7775	0Y7740			
Y7776	001000	JMP	*-2	complete loading
Y7777	0Y7774			

Set (P) = Y7770
(U) = 0
(A) = Y7777
(B) = Y4000
(X) = 0

If the formatted tape is on unit 1, set X = 1 and change locations Y7773 and Y7774 to 100011 and 101211, respectively.

Y = 2 for 12K memory, Y = 3 for 16K memory, etc.

CORE MAP



If DEBUG is eliminated, ($SYS) = Y6400 and tape buffers are allocated from this point.

Y = 2 for 12K memory, Y = 3 for 16K memory, etc.

B. Additions and Modifications to DEBUG Package and Magnetic-Tape Routines

1. Introduction

A number of additions and modifications have been incorporated into DEBUG and into the magnetic-tape routines. The following report provides information on these changes. The next two sections detail the new features in DEBUG and the modifications of the original DEBUG commands, with Section IV giving an updated summary of all the DEBUG commands. Section V describes the modifications to tape routines; to the user, these modifications appear minor, but they were necessary in order to accommodate four tape drives, and they result in a more flexible system. The next section describes the subroutine $DIV which compensates for some hardware shortcomings of the Varian 620/i DIVIDE instruction. Sections VII and VIII provide a revised Core Map and a revised System Loading Procedure.

2. New DEBUG Features

DEBUG may be operated from the CRT terminal or from the TTY. When DEBUG is re-loaded from the system tape, operation always begins on the TTY. The following command enables the user to switch between the terminals.

Command: < ("less than" sign)

Action:

Switch operation of DEBUG to the other terminal. The message "SWITCH" will be typed on the original terminal, and "CONTINUE" will be typed on the new terminal. This command may be given only at the left margin - i.e., only with all registers closed and with no arguments entered.

When the cursor reaches the bottom of the CRT screen, the user should erase the screen and re-position the cursor (by using the PAGE key, or the HOME and ERASE keys). The Q-command from the CRT (to restart DEBUG) erases the screen.

The following command can be helpful in debugging a program since it sets the memory to a known state before loading programs.

Command: = (for "Eqials 0")

Action:

Store zero in all non-system memory locations (cf. Core Map given in Section VII). This command may be given only at the left margin.

The following command enables a user to intersperse pertinent comments with TTY or CRT output.

Command: * (asterisk)

Action:

Set DEBUG to accept a comment: DEBUG will echo back all characters typed after the *, but it will take no action on the characters. A carriage return restores DEBUG to its normal mode of operation. This command may be given only at the left margin.

The following command allows a block of core to be displayed compactly on the CRT.

Command: G (for "Garbage")

Action:

Display in octal on the CRT the block of registers from Argl (mod 100_8) to Arg2 inclusive. 400_8 registers are displayed on one "page," and the PAGE key on the CRT terminal is used to display the next 400_8 registers and to return to DEBUG after the final page is examined. The display may be terminated prematurely be enabling Sense Switch 1 and using the PAGE key. All registers must be closed when this command is issued.

Example:

1111,2222,G	**Display registers 1111-2222 inclusive. The first page will actually show registers 1100-1477, the second page 1500-2077, and the final page 2100-2477.**

The contents of registers may be displayed as floating-point numbers or as decimal numbers with a specified binary point. The following two commands enable a user to select either of these modes. Information on the format of typed numbers is given in the following section. (The Varian 620/i manual gives details on the internal representation of floating-point numbers.)

Command: E (for "Exponent")

Action:

Set the mode to floating point. This command may be issued only if all registers are closed.

Command: Z (for Z register)

Action:

Open and show the contents of the Z register - the binary-point register. An error occurs if a register is already open.

The contents of the Z register represents the position of the binary point of registers displayed in the decimal mode. If (Z) = 0, the binary point is assumed to lie to the right of bit 0 and the value of the register will be interpreted as an integer; in general, if (Z) = k, for k positive or negative, the binary point is assumed to lie to the right of bit k and the value is interpreted as an integer divided by 2^k (equivalent to the data format Bk as interpreted by the Assembler).

The Z register is always displayed as a decimal integer. A carriage return closes the register and restores DEBUG to its former mode.

Example:*

0100/1000	Register 100 contains the value 1000_8.
Z/0.	(Z) = 0.
.100/512.	$1000_8 = 512_{10}$.
Z/0. 2	Set Z to 2.
100/128.0	1000_8 B2 = $512_{10}/2^2 = 128_{10}$
Z/2. -2	
100/2048.	1000_8 B-2 = $512_{10}(2^2) = 2048_{10}$

In order to facilitate communication between the user and the computer during program execution, a special entry point, $PAUSE, has been created in DEBUG. With the entry $PAUSE, a programmer can conveniently have a message typed while entering DEBUG to permit parameters to be entered or altered. Specifications for $PAUSE and an example of its usage are given below.

Calling Sequence: CALL $PAUSE, (MESSAGE ADDRESS)
where (MESSAGE ADDRESS) is the address of the text to be typed; the text must be terminated by a location containing 0.

Entry Parameters: None

Return Parameters: None

Registers Used: None

Remarks: The $PAUSE entry to DEBUG saves registers, types the text contained at MESSAGE ADDRESS, types a carriage return, and then allows the user to perform any of the DEBUG operations. To resume program execution, the user types C (for "Continue").

The symbol $PAUSE is recognized by the assembler.

*Underlined characters are those typed by DEBUG.

Example:

```
  *In this program when the first call to $PAUSE is reached, MSG1
  *will be typed.  The user then enters the desired value of PARAM
  *(in the appropriate mode), and types C to continue execution.
  *When computations are finished, MSG2 will be typed and, after
  *examining the results, the user may type C to rerun the program or
  *Q to terminate execution and restart DEBUG.

 AGAIN  NULL                             Start of program
          .
          .
          .
        CALL       $PAUSE,MSG1           Get parameter value
          .
          .
          .
        CALL       $PAUSE,MSG2
        JMP        AGAIN
        ORG        01000
PARAM   BSS        1
MSG1    DATA       'ENTER PARAM VALUE AT LOG 1000',0
MSG2    DATA       'DONE.  RESULTS AT LOC 2000',0
          .
          .
          .
```

3. Modifications to Original DEBUG Commands

The most significant changes to DEBUG have been to permit more flexibility in the display and in the entry of register values. The contents of registers may be examined in any of four modes: octal, decimal (with specified binary point), floating point,[†] or instruction. The mode may be changed only when all registers are closed. With all registers closed, input arguments are always octal values; with a register open, the input mode is the same as the output mode (except for instruction mode, in which

[†] The routine that displays numbers in decimal and floating formats was originally written by A. Griffiths.

case input is in octal). The following example illustrates this point:

O100/0 100	**Set register 100 to 100_8.**
.100/64. 100	**100_8=64_{10}; set register to 100_{10}.**
O100/144	**Register 100 now contains the octal value 144.**

Decimal and floating-point numbers are always typed with a decimal point by DEBUG. Decimal integers (i.e., (Z) = 0) are typed as full-precision integers (i.e., with no fractional part). Non-integer decimal values (i.e., (Z) $\neq$ 0) and all floating-point values are shown to four significant figures*, with exponential notation used if the value is less than 0.001 or greater than 9999. Decimal and floating-point arguments may be entered into DEBUG in either decimal or exponential notation. For example, all of the following input formats are equivalent:

1, 1.0, 1.00000, 1E0, 10E-1, 100.000E-2, .1E1, 0.999999

Note that it is not necessary to enter the decimal point.

The following minor modifications have also been incorporated in DEBUG:

- When any of the registers T, P, J, K, or N are opened, the mode is set to octal and remains octal after the register is closed. (However, when the value of P is shown by DEBUG at a trap - or after a program interrupt or a program step - the mode used is the current mode of DEBUG.) For example:

.A/100. 200	Set (A) to 200_{10}.
P/1000 100	P shown in octal and set to 100_8.
T/-1 101	Set trap.
N/4 1	Show only A after trap.
A/310	Mode is still octal.
.C	Set mode to decimal and continue execution.

*Zero is always typed as "0." .

P/65. A/200.	At trap, P and A are shown in decimal.
P/101	When open, P is shown in octal.

- If the space-bar is typed with a register open, DEBUG will display the contents of the register. This feature is useful in checking conversion accuracy. For example:

Z/0. 12	Set binary point to 12.
.1000/0. .49019 .4902;	With only 12-bit accuracy, the desired
1001/0. .62387 .6238	values are rounded as shown.
I100/ NOP 57777 STA* 777	Set 100 to the instruction, STA*,777, and check that the desired instruction is entered.

-The /-command cannot be given with a register open.

4. Summary of DEBUG Commands

Command:	Meaning
c.r. (carriage return)	Enter the last argument, close the open register (if any), and prepare to accept Arg1.
(space)	Enter the last argument; also, if register open, display contents of register.
*	Comment to be typed.
,	Enter the last argument and prepare to accept the next argument or command.
-	Argument is negative.
.	Set mode to decimal.
/	Show the contents of.
(0-9)	Digits for input arguments; only 0-7 for octal input.
:	Enter the last argument and open and show previous register.
;	Enter the last argument and open and show next register.
<	Switch operation to other terminal.

=	Set non-system core to 0.
A	A register.
B	B register.
C	Continue execution.
D	Dump a block of core.
E	Set mode to floating point.
F	I ind a masked value.
G	Display a block of core on CRT.
I	Set mode to "instruction".
J	J address (register to be typed after a trap).
K	K address (register to be typed after a trap).
L	Load from tape.
N	Number of registers to be typed after a trap.
O	Set mode to octal.
P	Program counter.
Q	Restart DEBUG.
R	Rewind a tape.
S	Instruction step.
T	Trap address.
V	Overflow indicator.
W	Write a value into memory.
X	X register.
Z	Binary-point register.
INT (special switch)	Program interrupt.
Any other character	Delete the last argument.

5. Modifications to Tape Routines

The tape routines have been re-written to handle up to four tape drives (numbered 0 - 3) from one controller[*]. The routines may be called even if one (or more) of the drives is offline: if an operation is attempted

[*] Bits 14 and 15 in the header code word specify the tape-drive number.

on an offline drive, the routine will return and indicate that a tape error has occurred.

The file-handling routines are used exactly as before. The tape-handling routines have been modified as follows:

$ECHK $DEV $UDEV	Deleted from system.
$CHK	Check tape status.
$CONN	Connect tape drive to controller.
$BIC	Sense BIC ready.

The three new routines are described below. The remaining tape-handling routines are used exactly as before; The new timing requirements for the tape-handling routines are given in the table at the end of this section.

$CHK

Check status of a tape unit after any tape operation.

Calling Sequence: CALL $CHK, EOT, EOF, ERROR

Entry Parameters: (X) = tape unit.

Return Parameters:

Normal return — In sequence: (A) = -4

Special returns — To EOT if end-of-tape encountered: (A) = -1

To EOF if end-of-file encountered: (A) = -2

To ERROR if any tape error:

- (A) = -3 if parity or write-ring error;
- (A) = -5 if tape is offline;
- (A) = Final memory location (≥ 0) if operation terminated prematurely (e.g., last location filled by $READ).

Registers Used: A

Timing. See table.

$CONN	**Logically connect controller to a tape unit.**
Calling Sequence:	CALL $CONN
Entry Parameters:	(X) = Tape unit.
Return Parameters:	None
Registers Used:	None
Timing:	See table.
Remarks:	All tape-handling routines - except $UNIT - call $CONN. Before actually connecting the controller to the specified unit, $CONN senses and saves the error conditions of the drive currently connected.

$BIC	Sense BIC ready.
Calling Sequence:	CALL $BIC
Entry Parameters:	None
Return Parameters:	None
Registers Used:	None
Timing:	See table.

Timing of Tape-Handling Routines

Routine	Minimum No. Cycles
$READ	85.75
$WRITE	85.75
$FMRK	77.75
$CHK	132.25 (normal return)
$CONN	73.75
$UNIT	5.5
$BIC	5.25

6. Divide Subroutine

The divide instruction on the Varian 620/i will produce misleading results if the dividend (numerator) is negative and an integral multiple of the divisor (denominator). The routine described below*, $DIV, compensates for the hardware shortcomings. and in addition provides a double-precision quotient. If, however, the numerator is known to be positive, and if only single-precision results are required, then $DIV need not be called.

$DIV — Divide routine that compensates for Varian 620/i hardware ideosyncracies.

Calling Sequence: CALL $DIV, DIVISOR
where DIVISOR contains the denominator.

Entry Parameters: (A), (B) = Numerator, double precision

Return Parameters: (A), (B) = Quotient, double precision B15 (i.e., the binary point is located to the right of A-register bit 0).
OF = Set if overflow occurred
= Reset otherwise

Registers Used: A, B, OF

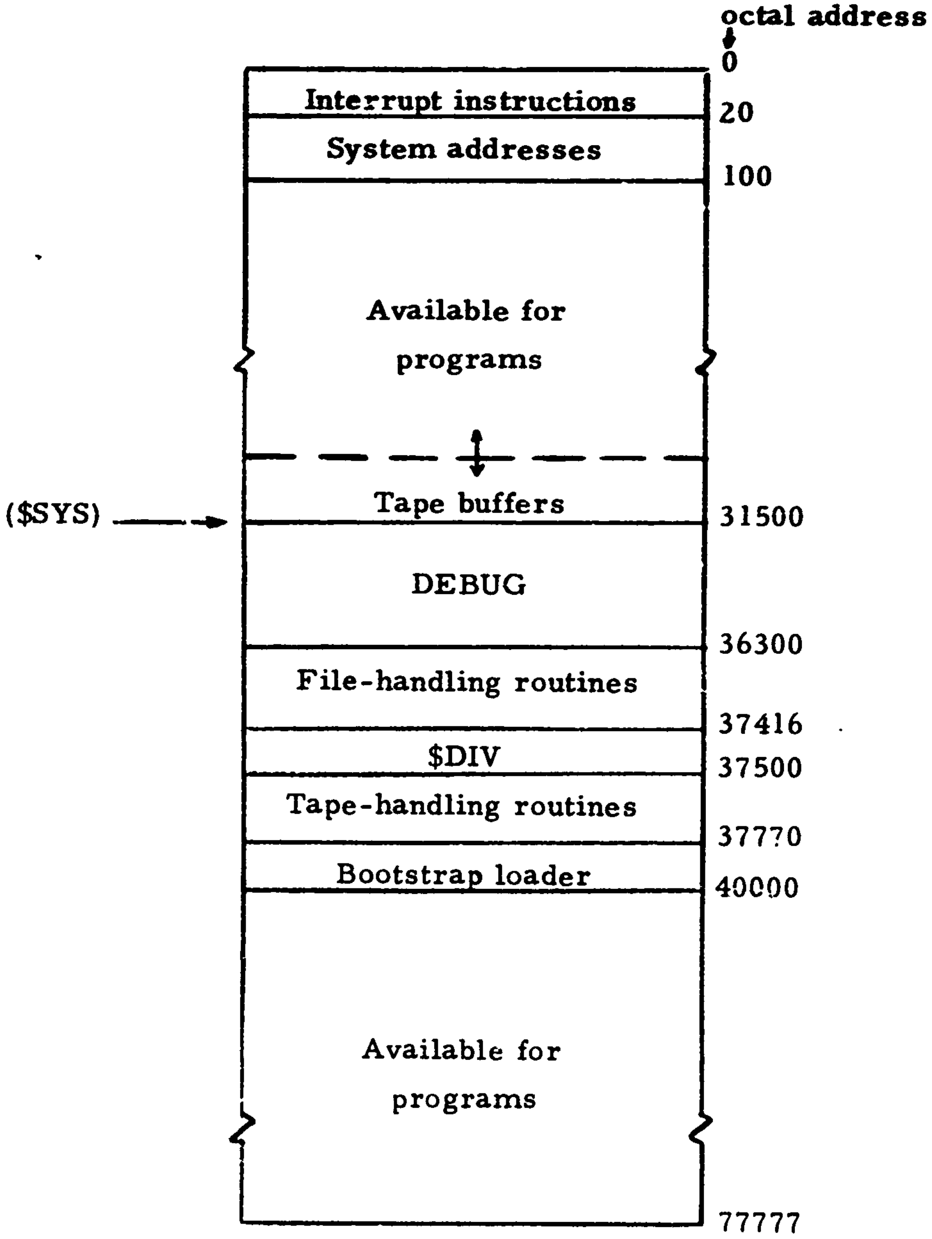
Timing: Varies from 51 cycles to 75 cycles depending upon sign and magnitude of numerator and denominator. Average time ~ 67 cycles.

Remarks: If (numer./denom.) ≥ 2^{15} or if (numer./denom.) < - 2^{15}, overflow occurs and the resulting quotient is meaningless.

The symbol $DIV is recognized by the assembler.

* This routine was originally written by C. Cappello.

7. CORE MAP



Because of shortcomings in the Varian hardware, the tape buffers cannot be allocated in locations above 40000. Memory that is available for programs is therefore split into two blocks.

If DEBUG is eliminated, ($SYS) = 36300 and tape buffers are allocated from this point.

8. System Loading Procedure

1) If the file-handling routines are intact:

Set	**P/37740**
	X/3*
	U/0
Turn	**TTY on**
Press	**SYSTEM RESET**
	Run

---

2) **If the Bootstrap Loader is intact:**

Set	P/37770
	A/40000
	B/31500
	X/3*
	U/104410*
	Position the system tape at the load point.
Turn	TTY on
Press	SYSTEM RESET
	RUN

---

3) To enter Bootstrap Loader:

Enable	REPEAT	
Set	U/54000	
	P/37767	
	A/bootstrap instruction	Repeat these operations until all bootstrap instructions are entered
Press	STEP	

Bootstrap Loader:

```
37767/0          37774/101210
37770/103220     37775/37740
37771/103121     37776/1000
37772/100020     37777/37774
37773/100010
```

*If the system tape is not on tape drive 3, then set X to the drive number of system tape. In this case, when using the Bootstrap Loader, set U to 104z10, where z = 1 + drive number.

C. Text Editor

Text Editor for the Varian 620/i

The Text Editor facilitates the preparation and modification of source-language programs or any body of text. The format and the repertoire of commands are similar to those on the CMS editor.

In general, three files of text are in use during editing: an input file from magnetic tape, a core file that is kept in memory, and an output file on magnetic tape. (If all text input originates from the teletype, then of course no input file is used.) The core file comprises the actual text that is available for examination and modification. The user should imagine a "pointer" marking his position in this file. Editing is done on a line basis, the reference point being the current position of the pointer. Added to the user's text are two null lines marking the top and the bottom of the file. The line pointer may be positioned at the null top line (represented by "T"), but this line cannot be delted, replaced, or moved. The pointer cannot be positioned at the null bottom line; if the bottom of the file is encountered, "B" is typed, and the pointer is set to the last text line.

Instructions to the Editor are given in one of the three forms

(i) C

(ii) Cn

(iii) C<u>string</u>

where C represents a command, n is a positive integer, and string represents a sequence of characters. Most commands are given by a single character; however, in order to minimize the possibility of a serious error, two-character commands are used for restarting the Editor and for exiting from the Editor. The integer n, if not specified, is assigned the default value n = 1, and the symbol "#" is interpreted as n = ∞. (For example, L# means in effect "load as many lines as possible.") If an error is made in typing an instruction, the character "@" may be used to delete the previous character, or the character "←" may be used to delete the entire line. (Two successive @ @'s delete the last two characters, etc.) An instruction is terminated when a carriage return is entered. The Editor then either executes the instruction or issues an error message. It is possible that instructions which add text to the core file (e.g., Input, Load) could result in an overflow of the memory area available to the Editor. If this situation occurs, the message "-FULL-" is typed and the instruction is not completed; the specific action taken by the Editor is detailed below with the command descriptions. When the editing of a body of text is completed, the output file must be closed (i.e., an end-of-file mark must be written on the tape). As described below, this is accomplished either with a Close command or with a Quit command.

Given now are descriptions of the Editor commands and error messages, and an example illustrating the use of most of the commands.

Commands

B -(Bottom) Move the pointer to the bottom of the core file.

C# -(Close) Store the remaining text on the output file, close the output file, and restart the Editor. If an output file has not yet been specified, the message "T, F =" will be typed by the Editor (after the user types a carriage return). The user should then enter the tape unit, file number of the output file.

Dn -(Delete) Delete n lines, starting with the current line. The pointer is set to the line preceeding the first deleted line so that text may then be inserted to replace the deleted lines. The null top line T is ignored by the Delete command. (For example, with the pointer at T, the instruction "D5" will delete the 5 lines following T.) If the bottom is encountered, "B" is typed.

E -(End-of-File) Terminate input from the current input file. (A new input file will be requested with the next Load command.)

Fstring -(Find) Starting with the next line, search for string and type the line if string is found. If not found, B is typed.

Hstring -(Here) After the current line, insert the lines specified by the last Move command (see M). If string is blank, the moved lines are left intact at their original location; if string is non-blank, the lines are deleted from their original location. The pointer is reset to the line where the move originated: the first moved line if the original lines were not deleted; the line preceeding the moved lines if the original were deleted. [An error message will be issued if a move has not been specified or if a text-altering command was executed between the Move and Here commands (see M). If a FULL message is given, the Move has been discarded.]

Istring -(Insert) Insert string after the current line. If string is blank, the Editor goes into the INPUT mode, and a number of lines may be inserted in succession without giving the I-command. If an empty line is typed in the INPUT mode, the EDIT mode is re-entered.

Ln -(Load) Load n lines of text from the input file, adding these lines at the bottom of the core file. The pointer is set to the first line of the new (just-loaded) text, and this line is typed out. If the last line of the input file has been loaded, the message "-EOF-" is also typed. As explained for Close, if the input file is not specified, the Editor will request one. A FULL message after a load command means that the space available for the core file has been exhausted (and consequently fewer than n lines have been loaded).

Mn -(Move) Prepare to move n lines (starting with the current line). Following the Move command the pointer should be positioned at the destination, and a Here command executed. Between the Move and the Here commands, no text-altering commands (Insert, Delete, etc.) may take place, or the Move information will be discarded. As with Delete, the null top line T is ignored by Move.

Nn -(Next) Position the pointer n lines forward from the current line.

Pn -(Print) Type n lines starting with the next line. If however, the current line is the last line, then this line is typed. The pointer remains at the last line typed.

Q# -(Quit) Store the remaining text on the output file, close the output file, and return to DEBUG. As with Close, an output-file identification will be requested if none has been specified.

Rstring -(Replace) Replace the current line with string. [An error message will be issued if string is blank. If a FULL message is given, the original line has been deleted, but the replacement text has been discarded.]

Sn -(Store) Store the top n lines on the output file and delete these lines from the core file. The pointer is set to the new top line, and this line is typed out. As with Close, an output-file identification will be requested if none has been specified.

T -(Top) Set the pointer to the top of the core file.

Un -(Up) Move the pointer n lines back from the current line.

Error Messages

? ? Invalid command,or invalid tape unit or file number.

-FILE- The specified input file is not a text file.

-ERROR-(TEXT LOST)

This message is issued if a non-fatal error (an error from which the system can recover) occurs during a tape read - i. e. after a Load command. The lost text is replaced by the characters '##', and loading continues until the specified number of lines have been loaded (or until end-of-file is reached). The error message is issued and the characters '##' are inserted for each block of text that is lost. After the Load command has been completed the user must re-type the lost text. (He may, of course, re-attempt the loading.)

-ERROR-(FILE CLOSED)

This message is issued if a fatal (non-recoverable) tape error occurs. If the error occurs during a read operation (Load), the input file has been terminated. If the error occurs during a write operation (Store, Close, or Quit), any text already on the output file is lost.

Example

Suppose we have two files on magnetic tape, one with the text

```
1
1
2
3
11
12
13
```

and the other with the text

```
4
5
7
7
8 .
```

We wish to generate a file with the integers 1-12, with one integer per line. In the following, underlined characters are those typed by the Editor. A carriage-return/line-feed follows the last character for each line. (The user need only type the carriage return; the Editor performs the line feed.)

The Editor is loaded, and the execution begins:

Teletype output	Contents of core file after execution of instruction	Remarks
EDIT:	(For convenience commas are used as line delineators)	
L6 (car. ret.) T, F=0, 1 1	1, 1, 2, 3, 11, 12	Load 6 lines from file 1, tape unit 0. ("1" was the first line loaded.)

Teletype output	Contents of core file after execution of instruction	Remarks
E		Terminate the input file
L#(car. ret.)T, F=0, 2 -EOF- 4	1,1,2,3,11,12,4,5,7,7,8	Load the entire contents of unit 0, file 2. End-of-file was reached on the input file. ("4" was the first line loaded.)
M5		Prepare to move last 5 lines.
U3 3		Set the pointer up 3 lines.
HX 12 B	1,1,2,3,4,5,7,7,8,11,12	Move the lines here and delete them from original location. (Pointer is set to "12", the line preceeding the first moved line.)
T		Set pointer to top
D	1,2,3,4,5,7,7,8,11,12	Delete the first "1"
F7 7		Find the first line containing "7".
R ? ?		Replace the line. Error. (No replacement text specified.)
R8@ 6	1,2,3,4,5,6,7,8,11,12	A typing error ("8") was deleted, and the "7" replaced by a "6".
N2 8		Forward 2 lines.
I INPUT:		Begin INPUT mode.
9	1,2,3,4,5,6,7,8,9,11,12	Insert "9".
10	1,2,3,4,5,6,7,8,9,10,11,12	Insert "10".
(car. ret.) EDIT:		Type empty line to return to EDIT mode.

Teletype output	Contents of core file after execution of instruction	Remarks
T	1,2,3,4,5,6,7,8,9,10,11,12	Top of file.
P#		Print everything.
1		
2		
3		
.		
.		
.		
11		
12		
B		
C#(car. ret.)T, F=1, 3	(empty)	Store the text on unit 1, file 3, close this file, and restart.
EDIT:		

D. Additional Command for Text Editor - 'o'

The following command has been added to the Text Editor on the Varian 620/i so that blocks of text may be read in from paper tape that has been prepared offline:

Command:

O - (Offline Paper Tape) Load a block of text from paper tape, inserting the text after the current line. (A blank frame of tape terminates a block of text.) The pointer is set to the first line of the new (just-loaded) text, and this line is typed out. [If a FULL message is issued the paper tape is stopped after the first line that could not be loaded.]

In the offline preparation of paper tape, the special characters "@" (for deleting the previous character) and "←" (for deleting an entire line) may be used in the same manner as in online text editing. Each line of text is terminated by a carriage return or by a line feed. If a line has more than 72 characters, it will be truncated to 72 characters. Empty lines are ignored.

E. Additional Command for Text Editor - 'x'

The following command has been added to the Varian text editors (CRT and TTY) to allow modifications to be made on individual lines of text:

Command:

Xd string 1 d string 2d - (Exchange)-In the current line of text replace string 1 with string 2. "d" is delimiter and may be any character not in string 1 or string 2 (except "@" and "←"). The third delimiter is not required; the command will execute properly if terminated solely by a carriage return. [An error message will be issued if string 1 is not found or if the resultant line is longer than 80 characters or blank.]

Examples:

Text	1234555
Command	X/555/567/C.R.
New Text	1234567
Command	X/67//C.R.
New Text	12345
Command	X/3/3AB/C.R.
New Text	123AB45
Command	X//AAA/C.R.
New Text	AAA123AB45

F. Differences Between the Varian Supplied Assembler and the Version Used at MIT/LL

1. Method of Operation

The Lincoln Laboratory version of the assembler supplied by Varian Data Machines is a two pass assembler operating with magnetic tapes and a CRT or teletype. The first time the source code is read in, locations are assigned to the symbols and certain errors are detected. On pass 2 the object or binary code is produced and a listing of the program is written on a magnetic tape suitable for printing on IBM equipment.

If the location assigned to a symbol on pass 2 does not agree with the location assigned on pass 1, a synchronous error results. (An example of how this can happen is an instruction of the form

```
BSS        N
```

where N is defined later in the program.) When such an error occurs, a message is printed and the user must enter either 'C' for continue the assembly or 'R' for restart the assembler.

2. Operating Pro~edure

The user must mount the tapes containing his source code and those on which he wishes his binary and listing output to go. The listing tape must be different from either the source or binary tapes which may be the same. Up to twenty files may be assembled into one program and these may be on different tapes.

Error messages are written out in the listing of the program and on either the teletype or CRT.Normally the crt is used, but the teletype may be selected by flipping on sense switch 3. Input parameters are also requested and entered using the device selected. At any time in an assembly the sense switch may be flipped and the next time input parameters are requested, it will be on the opposite device.

3. Input Parameters

The assembler requests the input parameters as it wants them. It will first ask for the source tape and file numbers. This list is entered as one tape number, comma, file number, carriage return per line. A 'Q' in place of the tape number returns the user to DEBUG. A '?' negates the entire argument line being entered. A null line (i.e., just a carriage return) terminates the list of source files. The binary and listing tapes and files are then requested. An 'R' may be typed in place of any tape number and the entire argument list will be requested again. An 'S' typed in place of the binary or listing tape number suppresses that output file. If several listings are to be done at a time, 'N' may be typed for the listing tape argument after the first time and the output will be put on the next file of the listing tape.

4. Source Statement Format

Each line of source code consists of label, instruction, and variable fields. (The remarks field is optional.) These fields may be of any length with a tab used to separate them. If there is no variable field required by the instruction, there still must be a tab after the op-code.

5. Binary Point Format

In addition to the forms Varian allows the variable field to take, the LL version of the assembler allows for a binary point format. In this format a decimal number may be entered with the position of the binary point specified. The result is a single word and the programmer must know where the binary point is.

The way to write a number in binary point format is

$$\underline{)}\ x \,\underline{.}\, y\ \underline{B} \pm N \ .$$

The underlined quantities are required while all others are optional. 'x . y' is the decimal fraction. 'N' specifies the position of the binary point. For 'N' equal to zero the result is the floating point number rounded to an integer.

If 'N' is so large that high order bits would be lost (overflow), an error message is typed and a full word of zeros assembled.

The following error messages could occur if the format is not correct.

Message	Cause	Result
*FA	no .	2 zero words assembled
*AD	no)	1 zero word assembled
*BA	overflow	1 zero word assembled

6. Shift Instructions

The mnemonics for four shift instructions have been changed for greater consistency. They are

LLSR	to	LSRL
LLRL	to	LRLL
LASR	to	ASRL
LASL	to	ASLL

All shift instruction abbreviations are now of the form ABCD where A indicates type (logical or arithmetic), B indicates operation (shift or rotate), C indicates direction (right or left), and D indicates register (A, B, or long).

7. Pseudo-Op Changes

A. MORE It no longer exists since there is no need for it.

B. STRT This pseudo-op has been added to allow a programmer to specify at any point in his program the entry point. Its usage is identical to that of the end instruction, but it does not terminate the assembly. Its value is in assemblying several different files into one program.

The entry point specified by the first STRT card takes precedence over an END card or any other STRT card. If more than one STRT card is used the error message *SF is typed. If neither an END nor a STRT instruction is used, the entry point is set to 100_8 and an appropriate message is typed.

C. END The END instruction is no longer required at the end of the program. The list of source inputs is processed until the end of the last file. If an END card is present, however, it terminates the assembly where it occurs.

8. Error Handling

Programming error messages are described on page 299 of the manual provided by Varian Data Machines. Errors that occur during the operation of the assembler such as tape errors result in self-explanatory error messages being typed out. Tape errors close all files that have been opened during the assembly; the assembler then returns to ask for new input parameters. (If a write error occurs, check to see that the tape has a write ring.)

9. Symbol Table

The assembler automatically puts in the symbol table of every program a list of commonly used system locations. These locations have a symbol beginning with a '$' associated with them. Thus a programmer may use these symbols in his program without defining them or caring where system routines are located. (To avoid conflict, use of symbols beginning with '$' should be avoided). These symbols will be printed in the symbol table of every program.

G. Tape Splice

Purpose: To copy a number of formatted files - in any order - from one magnetic tape to another. (Formatted files are those written with calls to $OPEN, $DATA, and $CLOSE and include text files written by the Editor and binary files written by the Assembler.)

Programmer and Date: I. Richer, May 1970

Input Parameters: Tape and file numbers entered via keyboard.

Sense Switch Settings:
SS3 - OFF for operation from CRT terminal
ON for operation from teletype.

Locations Used: 100_8 - 777_8

Usage: In response to messages written on the CRT screen (or on the teletype if SS3 is set), the user enters the following <u>octal</u> parameters:

OUTPUT TAPE	(0-3)
INPUT TAPE	(0-3)
1st OUTPUT FILE	
INPUT FILES:	

In response to the INPUT FILES message, the user enters a list of files in any order. Each parameter (including each input file number) is terminated by a carriage return. The input-file list is terminated by a blank line - i.e. by a line containing only a carriage return. At any point, the character 'Q' returns the user to DEBUG; 'R' rewinds the tapes and restarts the program. If an invalid character (or an invalid tape number),is entered a '?' is typed by the program, and the parameter must be re-entered.

After the final carriage return, the INPUT FILES are copied from the INPUT TAPE to the OUTPUT TAPE , with the first file on the list copied onto the 1st OUTPUT FILE, and the remaining files copied in sequence. After each file is copied, the output tape is backspaced and the file is verified (re-read). If a tape error is encountered, copying is halted and the appropriate error message is typed. When copying is complete, the program types the number of the next output file and requests more input files for copying between the same input and output tapes. The user then may enter another list of input files, may restart the program (e.g., in order to re-specify the input or output tape),or may return to DEBUG.

Error Messages:

?

-Invalid character (parameter must be re-entered)

$$\left\{\begin{array}{l}\text{READ}\\ \text{WRITE}\\ \text{VERIFICATION}\end{array}\right\}\quad \text{ERROR-COPYING HALTED AT INPUT FILE N}$$

-Tape errors encountered while copying file number N.

H. Tape Duplicate

Purpose: To copy, record-for-record, a number of files from one magnetic tape to another.

Programmer & Date: I. Richer, May 1970.

Input Parameters: Tape and file numbers entered via keyboard.

Sense Switch Settings: SS1 - OFF for normal operation
ON to terminate copying.

SS3 - OFF for operation from CRT terminal
ON for operation from teletype.

Locations Used: 100_8 - 31000_8

Usage: In response to messages written on the CRT screen (or on the teletype if SS3 is set), the user enters the following octal parameters:

OUTPUT TAPE	(0 - 3)
INPUT TAPE	(0 - 3)
FIRST FILE	
LAST FILE	

The character 'Q' returns the user to DEBUG. If an invalid character (or an invalid tape number) is typed, the OUTPUT TAPE will again be requested, and all input parameters must be re-entered. Each parameter is terminated by a carriage return.

After the final carriage return, the program copies, record-for-record, all files from the FIRST FILE through the LAST FILE, inclusive, from the INPUT TAPE to the OUTPUT TAPE. After each record is written the output tape is backspaced and the record is verified (re-read). If a tape error is encountered-either on reading, on writing, or on verifying - the operation is re-tried. If four errors occur in the same record, an error message is typed, but copying continues. SS1 may be set to terminate the copying. When copying is complete, the tapes are rewound.

Error Messages:

??

-Invalid character (the first input parameter will then be requested)

$\left\{\begin{array}{l}\text{READ}\\ \text{WRITE}\\ \text{VERIFICATION}\end{array}\right\}$ ERROR - FILE N

-Four tape errors were made in copying a record in file number N.

I. Tape Copy

Purpose: To copy either a paper tape file or a formatted magnetic tape file onto either a paper tape file or a formatted magnetic tape file. (Formatted files are those written with calls to $OPEN, $DATA, and $CLOSE and include text files written by the Editor and binary files written by the Assembler.)

Programmer and Date: I. Richer, May 1970

Input Parameters: Input medium and output medium (magnetic or paper tape). If magnetic tape, tape and file numbers.

Usage: In response to messages written on the teletype, the user enters the letter m to designate magnetic tape or the letter p to designate paper tape. If magnetic tape is specified, the program requests tape (0-3) and file number which are entered separated by a comma and terminated with a carriage return. At any point, the character 'Q' returns the user to DEBUG. If an invalid character (or an invalid tape number) is entered, a '?' is typed by the program, and the parameter must be re-entered. If a tape error is encountered, copying is halted and the appropriate error message is typed. When copying is complete, the user then may enter another input list, or may return to DEBUG.

Error Messages:

?

- Invalid character (parameter must be re-entered).

WRITE ERROR - COPY TERMINATED

READ ERROR - COPY TERMINATED

- Tape errors encountered while copying file.

J. List/Dump

Purpose: To copy a file on a Varian user tape to a file on another tape in such a format that it can be printed on the IBM 360/40.

Programmer: C. D. Cappello

Date: May 1969

Locations used: 100_8 - 1625_8

Input parameters:

1. The numbers of the output tape and file.
2. The numbers of the tape and file to be copied.

Usage: In response to messages written on the teletype the user enters the input parameters in octal. An 'N' in place of the output tape and file numbers specifies the next file on the specified tape to be used. A 'Q' returns the user to DEBUG.

Error messages:

? - An illegal character has been typed as an input value. Both the tape number and the file number must be entered again.

TAPE CHOICE - The input and output tapes must be different. Re-enter the input parameters.

FILE TYPE ILLEGAL - At present only file types of binary and text are accepted; hence, any other type illegal. Re-enter the input parameters.

RECORD LOST - A read error has occurred. Data is lost but the program continues.

UNRECOVERABLE READ ERROR - A read error has occurred and the program cannot continue reading. A file mark is written on the output tape and the program restarted.

WRITE ERROR - A write error occurred. A file mark is written and the program restarted.

NO WRITE RING - Check to see that a write ring is in and the write enable light on.

Remarks: The tape and file numbers of the tape to be copied are printed at the top of the output. Text files are written in a listing format and binary files are printed as an octal dump.

IV. GENERAL-PURPOSE SUBROUTINES

A. Plotting

1. INTRODUCTION

The plotting routines facilitate the display of data in graphical form on the Computek CRT terminal, which is interfaced as an I/O device with the Varian 620/i computer, and on the X-Y plotter, which serves as an offline facility. These routines are designed to eliminate as much as possible the tedious chores associated with graphing and still retain enough flexibility to accommodate a wide variety of graphs. With these routines the user need not concern himself with specific commands directed to the CRT; the programs will automatically:

a. Scale axes,

b. Position axes on the CRT screen,

c. Place either grid marks or grid lines on axes,

d. Label axes and title graphs,

e. Number axes limits,

f. Perform linear transformation of data coordinates to screen coordinates,

g. Plot data curves on grids.

The versatility of these routines is indicated by the options available to the user:

a. One set of axes may be displayed full screen or two sets of axes may be displayed with one in the top [illegible] and the other in the bottom half for direct comparison.

b. Axes limits may be specified or determin[illegible] the limits of data to be displayed.

c. Data points may be placed on the axes with or without lines joining the points.

d. Several sets of data may be displayed on the same axes, each set having points marked with a different character.

e. The x coordinates of data may be specified explicitly or implicitly in the form of an initial value and uniform increments along the x axis.

f. The point coordinates may be stored in a buffer and plotted consecutively or may be individually added to a set already displayed on the screen.

When it is desired to graph the contents of a buffer, an executive routine is available which already includes the appropriate calls to the plotting routines. The characteristics of this executive routine are:

a. Screen is initially erased.

b. Only one curve on a full screen graph is drawn.

c. Y coordinates consist of data values.

d. X coordinates are stepped by 1 from an initial value of 0.

e. All N points are plotted.

f. N-1 appears as upper limit of X.

g. Minimum value of data is lower Y limit.

h. Maximum value of data is upper Y limit.

i. Item name appears as ordinate label.

j. A horizontal line at Y=0 is drawn if MIN < 0 < MAX.

k. Ordinate numbers are divided by the scale in the Item Block. If 1 ≤ scale ≤ 32767; otherwise the numbers are not scaled and a message identifying the scale is written on the screen.

2. USAGE

Efficient programming and ease of user operation dictate that the input parameters be transferred to the plotting routines in block form with only addresses of the blocks passed as arguments of a CALL instruction. Since the input parameters fall naturally into two distinct categories - a set associated with the display of coordinates axes and a set associated with the display of data points - two separate block forms are used. These blocks, which must be in the user's program, are designated as an axis block, BAX, and a data block, BPLT, and are described in Appendix I. The advantage of using two separate block forms is that curves from several different data blocks can conveniently be placed on the axes described by one axis block. As a general rule, the user's blocks are left unchanged by the plotting subroutines; this rule is violated only in the following cases. Both PLOT and

PJOIN fill BPLT+7 and BPLT+8 with the last data point coordinates. PLIMIT fills BAX+2 and BAX+3 with extrema of Y coordinates of data and, if C(BPLT) $\neq$ 0, fills BAX and BAX+1 with extrema of X coordinates of data. Input information to the executive routine is a special block form described in Appendix II.

The plotting routines are used to display graphs on the CRT and to punch paper tape for drawing graphs on the X-Y plotter. For the latter use, PPPP SET 1 must be inserted into the main program at assembly time and both SS3 and the paper tape punch must be on at execution time; these conditions, however, do not affect graphs displayed on the CRT. The paper tape created will draw only data curves on the plotter. Axes for these curves are available on a separate paper tape punched using a special program for that purpose.

On the CRT screen, though, axes, labels, numbers, and titles appear with the curves. There are separate routines for displaying axes and for drawing curves; this provides the user with the flexibility of placing several curves on the same set of axes. However, it requires him to make a minimum of two separate calls to draw a complete graph. The user may select either one graph drawn full screen, top half screen, or bottom half screen. The last two choices are provided so that two separate graphs can be displayed at the same time for direct comparison. The screen is not automatically erased prior to the placement of an axis; the user must either insert the instruction CALL $DCLR in his program or type the page key or the erase key on the CRT keyboard in order to erase the screen.

If PPFU SET 1 is included in the user's program at assembly time, the graphs will be drawn full width of the screen. Otherwise, the graphs are offset to the right to allow room for printing out information with DEBUG. These routines occupy 948 memory locations anywhere in core. If paper tape output is desired so that PPPP SET 1 is in the user's program, these routines occupy 1033 memory locations. In order to reduce the possibility of a label conflict, all labels of the plotting routines begin with the letter P, while the I/O routines begin with the letters $D. The user is urged to avoid beginning any of his labels with those letters. No indirect pointers and no literals are used in order that these routines be fully compatible with any program. No error messages are included with these routines. The program will continue to interrogate the CRT until a ready state is sensed; only then will it send output to the CRT.

3. EXAMPLES

Outlined below are typical instruction sequences illustrating the usage of the plotting routines. Figures 1-4 are the actual graphs displayed by examples a-d.

(a)

```
     CALL      $DCLR            ERASE SCREEN
     CALL      PLIM,A1,D2       LET DATA DETERMINE GRAPH LIMITS
     CALL      PAX,A1           DRAW AXES
     LDA       =5               STEP INITIALLY FROM X=5
     STA       D2+2
     CALL      PLOT,A1,D2       PLOT 1ST 10 POINTS
     LDA       =60              STEP FROM X=60
     STA       D2+2
     CALL      PJOIN,A1,D2      JOIN CURVE ONTO ITSELF
     CALL      PHOR,A1,0        DRAW HORIZONTAL LINE AT Y=0
     CALL      PHOR,A1,60       DRAW HORIZONTAL LINE AT Y=60
     CALL      PMESG,600,770,MSG1
     LDA       =3
     CALL      PONV             OUTPUT NUMBER
     CALL      PMESG,600,750,MSG2
     LDA       =10
     CALL      PONV             OUTPUT NUMBER
     CALL      $PAUSE,=0        ENTRY TO DEBUG
       .
       .
       .
MSG1 DATA      'TAPE= ',0
MSG2 DATA      'FILE= ',0
A1   DATA      0,100,0,0,1,ABS,ORD,TITLE,1,1
D2   DATA      0,Y1,0,5,10,1,'+',0,0
Y1   DATA      0,-10,150,60,80,40,120,-20,-30,0
ABS  DATA      'ABSCISSA',0
ORD  DATA      'ORDINATE',0
TITLE DATA     'GRAPH TITLE',0
```

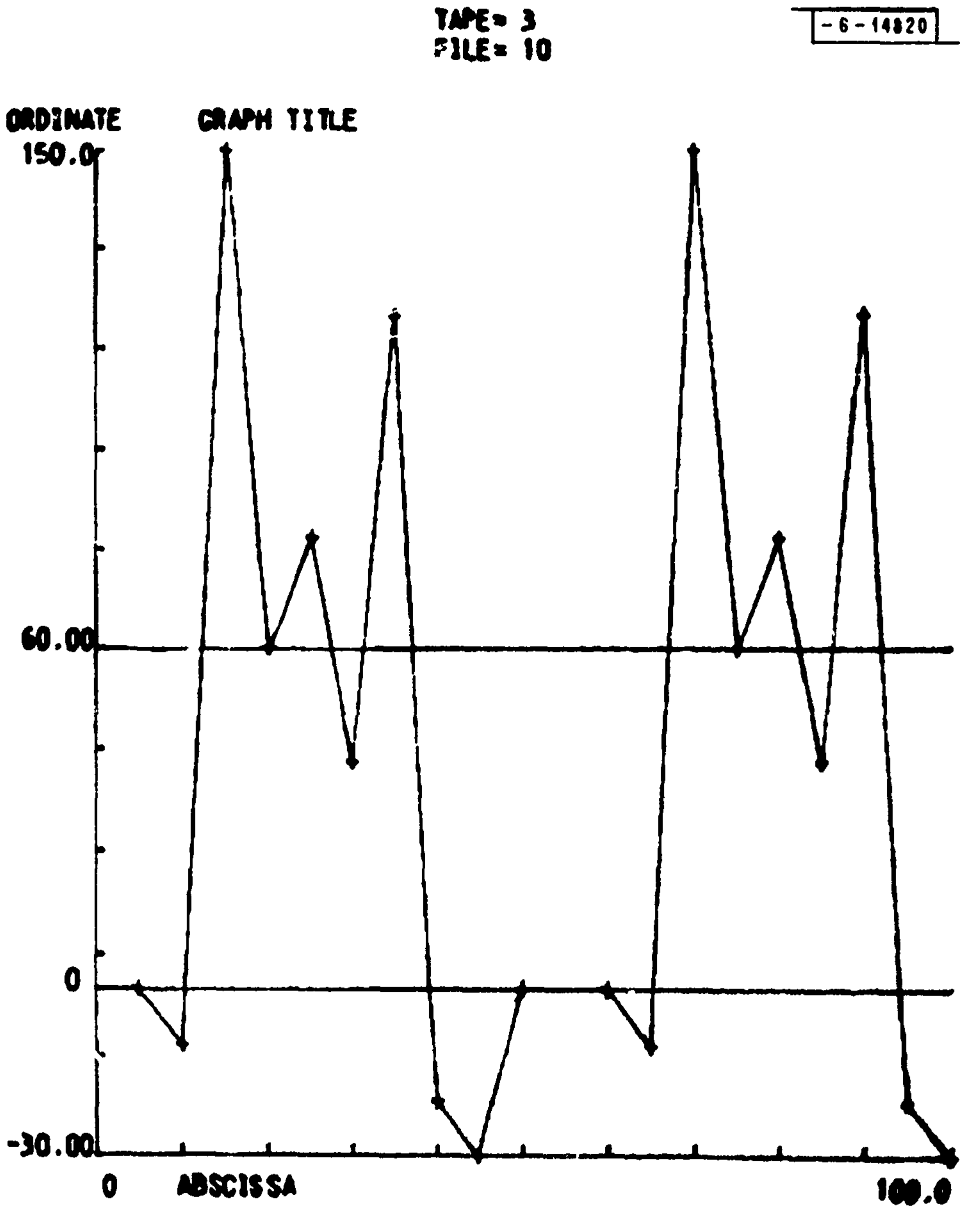


Fig. 1. Example (a).

(b)

```
        .
        .
        CALL      $DCLR            ERASE SCREEN
        CALL      PAX,A3           DRAW LOWER AXES
        CALL      PLOT,A3,D1       PLOT DATA
        CALL      PAX,A2           DRAW UPPER AXIS
        CALL      PLOT,A2,D1       PLOT DATA
        CALL      PINV,A2          READ CURSOR POSITION
        CALL      PINV,A2          READ CURSOR POSITION
        CALL      $PAUSE,=0        ENTRY TO DEBUG
          .
          .
          .
          .
  A2    DATA      -50,200,-50,200,2,ABS,ORD,TITLE,1,1
  A3    DATA      -50,200.-50,200,3,ABS,ORD,TITLE,1,1
  D1    DATA      X1,Y1,0,0,10,1,0,0,0
  X1    DATA      -20,10,20,30,40,50,60,70,80,150
  Y1    DATA      0,-10,150,60,80,40,120,-20,-30,0
  ABS   DATA      'ABSCISSA',0
  ORD   DATA      'ORDINATE',0
 TITLE  DATA      'GRAPH TITLE',0
```

(c)

```
          .
          .
          .
        CALL      $DCLR            ERASE SCREEN
        CALL      PAX,A4           DRAW AXES
        CALL      PLOT,A4,D4       PLOT DATA
        CALL      PMESG,850,770,CDATE    PRINT DATE
        CALL      PMESG,600,702,LABEL    PRINT IDENTIFYING LABEL
          .
          .
          .
  A4    DATA      0,255,)-40.B8,0,-1,ABS,ORD,0,1,)1.B8
  D4    DATA      0,Y4,0,1,256,1,0,0,0
  Y4    BSS       256
  ABS   DATA      'NO. OF PTS.',0
  ORD   DATA      'SPECTRUM (DB)',0
 CDATE  DATA      'AUG 24, 1972',0
 LABEL  DATA      'FREQUENCY 0-500 (HZ)',0
```

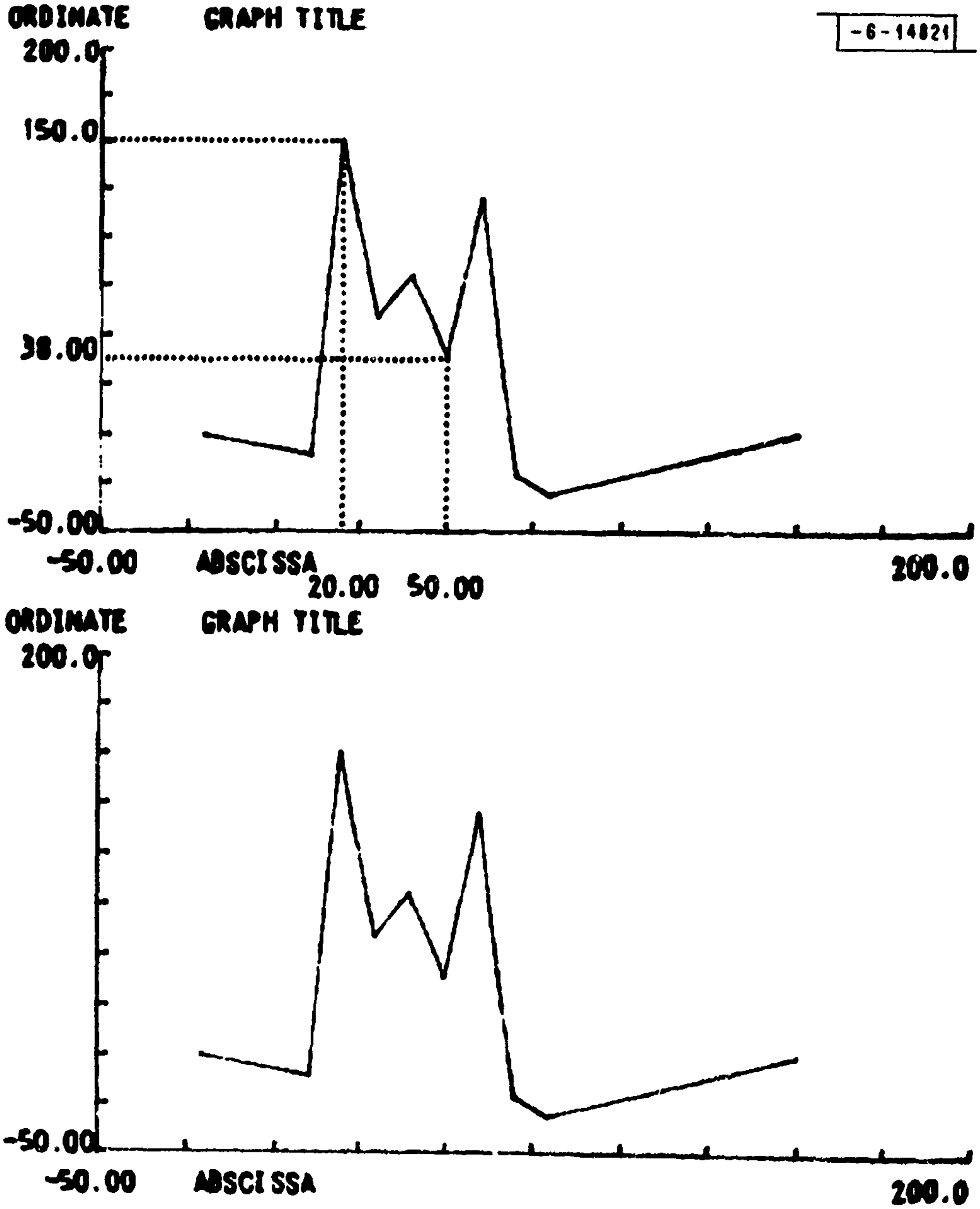


Fig. 2. Example (b).

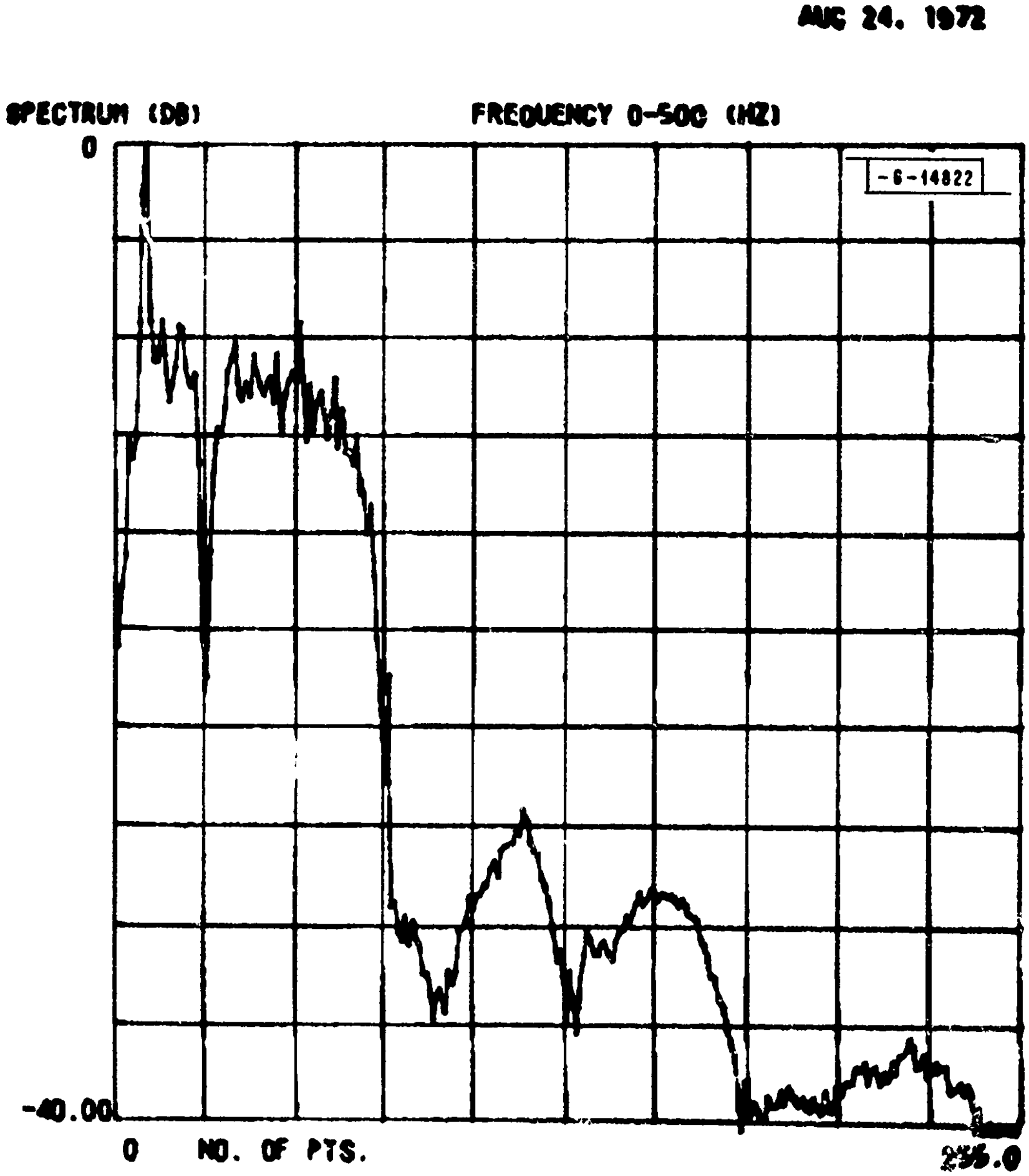


Fig. 3. Example (c).

(d) Prior to the execution of the following instructions, the buffer is filled with data representing an amplitude modulated carrier.

```
           .
           .
           .
      LDX       =ITEM              ITEM ADDRESS
      CALL      PG                 GRAPH BUFFER CONTENTS
      CALL      $PAUSE, =0         ENTRY TO DEBUG
           .
           .
           .
 ITEM DATA      'TEST', 01700, 161, 0, 0, BUFF
 BUFF BSS       161
CDATE DATA      'JUL 20, 1972', 0
```

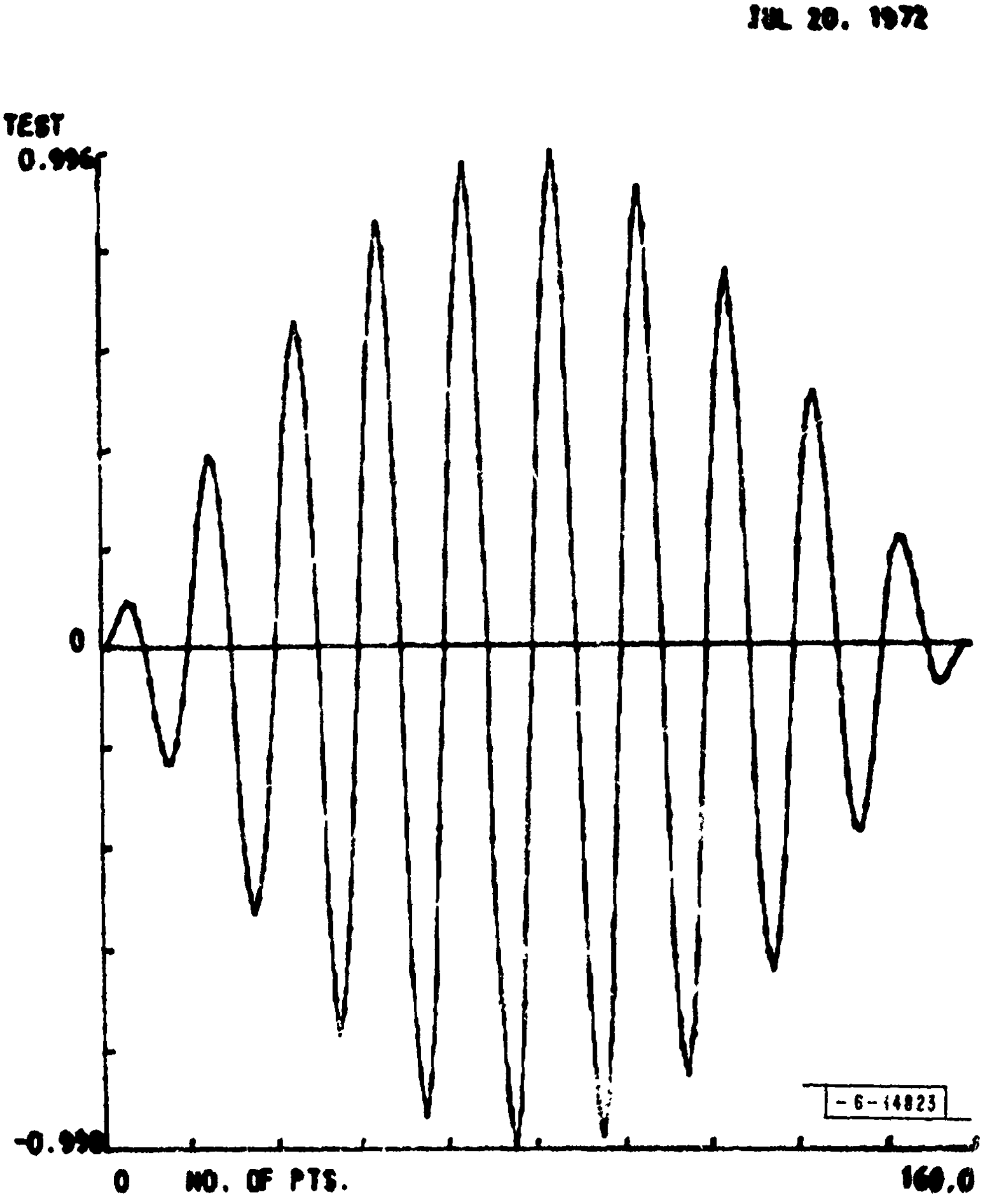


Fig. 4. Example (d).

PG — Executive subroutine to graph the contents of a buffer.

Calling Sequence: CALL PG

Input Parameters: C(X) = address of input block

Output Parameters: None

Registers Used: A, B, X, and OF

Remarks: A set of y coordinates stored in a buffer is plotted against a buffer index value on a set of axes drawn full screen, where the y axis limits are the extrema of the data values. A horizontal line at the value y=0 is drawn, provided it lies within the graph limits. The item name appears as the ordinate label. An identifying date or label at the upper right corner of the screen will be displayed if the user has included the ASCII code for this (terminated by a 0) with label CDATE; e.g., the user must supply: CDATE DATA 'JAN 1, 1972',0 . If location 75_8 contains the address of ITEM and if the address following the input block (ITEM+7) contains the address of PG, then typing @ ITEM$ on the I/O terminal in DEBUG environment will display the buffer on the CRT screen. The input block is described in Appendix II.

PAX	Subroutine to draw one set of axes on the CRT.
Calling Sequence:	CALL PAX, BAX
Input Parameters:	None
Output Parameters:	OF = 0
Registers Used:	OF
Remarks:	One set of axes specified in the axis block BAX is drawn either full screen, top half screen, or bottom half screen. Axes are offset to the right to leave room for I/O with the CPU. The screen is not initially erased; to erase the screen, user must either insert CALL $DCLR in his program or type page key or erase key on the CRT keyboard prior to drawing axes. User may specify either small tick marks on each axis or full grid lines; in either case each axis will be divided into 10 intervals. Each axis is numbered in decimal by its lower limit and its upper limit divided by the scale for that axis (scale ≥ 1) with the results truncated to 4 digits (5 if result > 9999). Graph title and labels for each axis may be provided. Labels and title texts are terminated by a zero word. About 60 character spaces are available for title and ordinate label. CRT cursor is returned to its original position after this routine is executed.

PLOT Subroutine to plot a set of data points on the CRT

Calling Sequence: CALL PLOT, BAX, BPLT

Input Parameters: None

Output Parameters: OF = 0

Registers Used: OF

Sense Switch Settings: SS3 on if paper tape output is desired in addition to CRT display.

Equipment: Paper tape punch on if paper tape output is desired in addition to CRT display.

Remarks: A set of N data points specified in the data block with pointer BPLT is displayed on the axes specified in the axis block with pointer BAX. The Y coordinates of data are stored in consecutive locations. The X coordinates of data either are stored in consecutive locations or are uniformly spaced from an initial value with a specified spacing. Points are marked with any specified symbol whose ASCII code is given in BPLT + 6. (Note that to mark a point with a dot, C(BPLT + 6) = 0, since a normal period does not center properly.) User may specify that adjacent plotted points be joined with straight lines. Y values of data outside the Y limits are replaced by values just beyond the nearest Y limits. Data points with X values outside the X limits are not plotted. Repeated executions of PLOT may be made after any execution of PAX, in order to place several curves on one set of axes. CRT cursor is returned to its original position after this routine is executed.

PJOIN Subroutine to plot a set of data points on the CRT and join this set with the previous set plotted from the same data block.

Calling Sequence: CALL PJOIN, BAX, BPLT

Input Parameters: None

Output Parameters: OF = 0

Registers Used: OF

Sense Switch Settings: SS3 on if paper tape output is desired in addition to CRT display.

Equipment: Paper tape punch on if paper tape output is desired in addition to CRT display.

Remarks: A set of N data points specified in the data block with pointer BPLT is displayed on the axes specified in the axis block with pointer BAX and is joined with the point whose coordinates are (C(BPLT + 7), C(BPLT + 8)). On exiting from this routine and routine PLOT, the coordinates of the last data point are stored in BPLT + 7 and BPLT + 8; hence, this routine enables the user to add on a set of points to an existing curve on the screen. The Y coordinates of data are stored in consecutive locations. The X coordinates of data either are stored in consecutive locations or are uniformly spaced from an initial value with a specified spacing. Points are marked with any specified symbol whose ASCII code is given in BPLT + 6. (Note that to mark a point with a dot, C(BPLT + 6) = 0, since a normal period does not center properly.) User may specify that adjacent plotted points be joined with straight lines. Y values of data outside the Y limits are replaced by values just beyond the nearest Y limits. Data points with X values outside the X limits are not plotted. Repeated executions of PJOIN may be made after any execution of PAX in order to draw a curve, one point or any other number of points at a time. Before the first use of PJOIN, C(BPLT + 7) and C(BLT + 8) must be initialized by either having prior use of PLOT with the same blocks or by storing an abscissa outside the X limits in BPLT + 7; in the latter case PLOT and PJOIN perform identically. CRT cursor is returned to its original position after this routine is executed.

PLIMIT — Subroutine to use data extrema for determining graph limits.

Calling Sequence: CALL PLIMIT, BAX, BPLT

Input Parameters: None

Output Parameters: OF = 0

Registers Used: OF

Remarks: Extrema of both X and Y data specified in the data block with pointer BPLT are stored as X and Y limits, respectively, in an axis block with pointer BAX. If X coordinates are uniformly spaced rather than explicitly stored, then X limits in the axis block remain unchanged. CRT cursor is returned to its original position after this routine is executed.

PHOR

Subroutine to draw a horizontal line across a graph at a given Y position.

Calling Sequence:

CALL PHOR, BAX, Y

Input Parameters:

None

Output Parameters:

OF = 1

Registers Used:

OF

Remarks:

Y is the data ordinate value specifying the position of the horizontal line which is to be drawn on a set of axes specified by the axis block with pointer BAX. The line is drawn only if Y lies between the lower Y limit and the upper Y limit specified in the axis block; otherwise no action is taken. When the line is drawn, it is numbered in decimal by the value Y divided by the scale for the Y axis (scale $\geq$ 1) with the results truncated to 4 digits (5 if result $>$ 9999). CRT cursor is returned to its original position after this routine is executed.

PINV — Subroutine to read cursor position coordinates, map to data coordinates, draw dotted lines from cursor position to axes, and print on the CRT the values of the data coordinates divided by axis scale.

Calling Sequence: CALL PINV, BAX

Input Parameters: None

Output Parameters:
C(A) = data abscissa equivalent of cursor location
C(B) = data ordinate equivalent of cursor location
OF = 0

Registers Used: A, B and OF

Usage: When PINV is executed, the CPU is initially waiting for any input from the CRT. During this time, the operator may position the cursor with cursor positioning keys on the keyboard. After the cursor is at the desired position, the operator types any alphanumeric key on the CRT. Dotted lines are then drawn from the cursor position to the axes, and the values of the data coordinates divided by the appropriate scale for each axis are displayed.

Remarks: Although the screen has a 10 bit resolution, the cursor position is read to only 8 bit resolution. Since graphs occupy something substantially less than full screen, overall accuracy of the cursor reading feature is about 1% of full scale. CRT cursor is returned to its original position after this routine is executed.

PMESG — Subroutine to write a message on the CRT at a specified location.

Calling Sequence: CALL PMESG, XCRT, YCRT, MESSAGE ADDRESS

Input Parameters: None

Output Parameters: None

Registers Used: None

Remarks: Message address is a pointer to text in ASCII code which is to be outputted. Text is terminated by a zero word. Coordinates (XCRT, YCRT), are the CRT coordinates specifying the location of the lower left corner of the initial character in the message. Restrictions on these coordinates are $0 \le XCRT \le 1011$, $0 \le YCRT \le 788$, where (0, 0) are the coordinates of the lower left corner of the screen. A single character lies in a cell of dimension 12 points wide and 20 points high; thus, there is room for 40 lines with 85 characters on each line. A frequent use of this routine is to identify a parameter associated with a graph on display. In that case a message, such as 'FREQUENCY = ', could be printed a little above the top of the graph. Then the value of that parameter should be loaded into the A register, and a CALL PONV executed to print the decimal value of that parameter on the screen. Screen coordinates locating corners of the graphs drawn are given in Table 1.

PWRT	Subroutine to write a message on the CRT at the cursor location.
Calling Sequence:	CALL PWRT
Input Parameters:	C(X) = MESSAGE ADDRESS
Output Parameters:	None
Registers Used:	A, X
Remarks:	Message address is a pointer to text in ASCII code which is to be outputted. Text is terminated by a zero word.

PONV — Subroutine to output decimal value of A register on the CRT at cursor location.

Calling Sequence: CALL PONV

Input Parameters: C(A) = number

Output Parameters: None

Registers Used: None

Remarks: A frequent use of this routine is to print the value of a parameter on the screen immediately after writing a message with PMESG which both identifies the parameter and positions the cursor to the next character cell following the message.

POSV — Subroutine to output the scaled value of the B register on the CRT at the cursor location (i.e., output = C(B)/C(A)).

Calling Sequence: CALL POSV

Input Parameters: C(A) = Scale
C(B) = Number

Output Parameters: None

Registers Used: A, B, X, OF

Remarks: A frequent use of this routine is to print the value of a non-integer parameter on the screen immediately after writing a message with PMESG which both identifies the parameter and positions the cursor to the next character cell following the message. The results are truncated to 4 digits (5 if result > 9999).

PEXTRM	Subroutine to determine extrema of a data set in consecutive memory locations.
Calling Sequence:	CALL PEXTRM
Input Parameters:	C(A) = pointer to data set C(X) = number of data words
Output Parameters:	C(A) = maximum of data set C(B) = minimum of data set C(X) = 0 OF = 0
Registers Used:	A, B, X and OF

PHSTOP — Subroutine to punch a character on paper tape which will stop the paper tape reader for the X-Y plotter.

Calling Sequence: JS3M PHSTOP

Input Parameters: None

Output Parameters: None

Registers Used: None

Remarks: A stop paper tape reader command between graphs is necessary in those situations in which the user wants to punch several graphs on a single paper tape and yet wants to display them on different charts. Also, without this command after the last graph, the paper tape will wind off entirely from the feed reel on the X-Y plotter.

APPENDIX I

Information blocks to the plotting subroutines have the following form, where BAX is a pointer to an axis block and BPLT is a pointer to a data block:

BAX+0: XL = Lower Bound on X Graphed

+1: XU = Upper Bound on X Graphed

+2: YL = Lower Bound on Y Graphed

+3: YU = Upper Bound on Y Graphed

+4: CODE = ± 1, One Graph Drawn Full Screen
= ± 2, Top Graph of Two Graphs Drawn
= ± 3, Bottom Graph of Two Graphs Drawn
+ Indicates Short Markers Will be Drawn on Each Coord. Axis
- Indicates Grid Lines Will be Drawn

+5: ABS = Address of Abscissa Label (Terminated by 0)
0 = Default on Label

+6: ORD = Address of Ordinate Label (Terminated by 0)
0 = Default on Label

+7: TITLE = Address of Title Text (Terminated by 0)
0 = Default on Title

+8: X SCALE = Scale on X Axis

+9: Y SCALE = Scale on Y Axis

BPLT+0: X ARRAY = Address of First X Coordinate
= 0, X Values Will be Stepped

+1: Y ARRAY = Address of First Y Coordinate

+2: XMIN = First X Value when Stepped (Ignored if not Stepped)

+3: DELTA = X Spacing when Stepped (Ignored if not Stepped)

+4: N = Number of Points to be Plotted

+5: ARG = 1, Lines Join Adjacent Plotted Points
= 0, No Lines Drawn

+6: 'SYMBOL' = ASCII Code of a Single Character to Mark Points
= 0, Point Marked with a Dot

+7: STORAGE (PLOT and PJOIN Insert X Coordinate of Last Point)

+8: STORAGE (PLOT and PJOIN Insert Y Coordinate of Last Point)

APPENDIX II

The information block to the executive plotting routine, PG, has the following form, where ITEM is the pointer to the input block:

ITEM+0: 'NAME' = 4 ASCII characters packed into 2 words to label ordinate

+2: FORMAT = position of binary point of data in 2's complement form in bits 15-6.

+3: N = buffer length

+4: Ignored

+5: Ignored

+6: ARRAY = buffer address.

The form of this input block is identical to the form of the input blocks to a receiver simulation program written on the Varian 620/i. For that reason there are some unused locations in the input block to the executive plotting routine.

TABLE 1

CRT Coordinates and Plotter Coordinates of Graph Corners

		Code	Lower Left Corner	Upper Right Corner
CRT	Full Width	1	80,47	1020,687
		2	80,447	1020,767
		3	80,47	1020,367
	Partial Width	1	320,47	1020,687
		2	320,447	1020,767
		3	320,47	1020,367
PLOTTER		1	0,0	5600,5120
		2	0,0	5600,5120
		3	0,0	5600,5120

B. Mean and Variance

Purpose: To compute mean and variance of sets of data in consecutive locations. An initialization subroutine, an execution subroutine, and a finalization subroutine are provided.

Programmer and Date: J. Michaud, July 1970

Calling Sequence:

```
CALL   #JIN, BLOCK      (initialization routine)
CALL   #JEX, BLOCK      (execution routine)
CALL   #JFI, BLOCK      (finalization routine)
```

where BLOCK is the starting address of a block of parameters.

Input Parameters:

C(BLOCK+0) = pointer to the data array

C(BLOCK+1) = m ≡ number of points in the array

Output Parameters:

From #JEX:

C(BLOCK+2) = n ≡ current total number of data points

$$\left.\begin{matrix}\text{C(BLOCK+3)}\\ \text{C(BLOCK+4)}\end{matrix}\right\} \sum_{i=1}^{n} \text{data (double-precision)}$$

$$\left.\begin{matrix}\text{C(BLOCK+5)}\\ \text{C(BLOCK+6)}\\ \text{C(BLOCK+7)}\end{matrix}\right\} \sum_{i=1}^{n} \text{data}^2 \text{ (triple-precision)}$$

OF set if $n > 32767 = 2^{15}-1$ (OF reset otherwise)

From #JFI:

C(BLOCK+2) = n ≡ total number of data points

$$\left.\begin{matrix}\text{C(BLOCK+3)}\\ \text{C(BLOCK+4)}\end{matrix}\right\} \text{mean (double precision, B15)}$$

C(BLOCK+5) = 0

$$\left.\begin{matrix}\text{C(BLOCK+6)}\\ \text{C(BLOCK+7)}\end{matrix}\right\} \text{variance (double-precision integer)}$$

Registers Used: OF used by #JEX

Locations Used: 355_8, anywhere in core.

Usage: Subroutine #JIN sets to zero locations BLOCK+2 through BLOCK+7.

Subroutine #JEX updates the current total of data word summed since the last call to #JIN, computes the runn sum Σ data, and computes the running sum Σ data^2.

Subroutine #JFI computes the mean and variance according to the following formulas:

$$\text{mean} = \frac{1}{n} \Sigma \text{ data}$$

$$\text{variance} = \frac{1}{n} [\Sigma\, \text{data}^2 - (\text{mean})(\Sigma\, \text{data})]$$

where the mean is in standard double-precision format (second word always positive) with the binary point to the right of the first word, and the variance is a double-precision integer.

Remarks:

No indirect pointers and no literals are used in order that these subroutines be fully compatible with any program.

Repeated calls to #JEX may be made with the same or different input parameters before calling #JFI.

The user should check OF on return from #JEX since erroneous results occur if $n > 32767$.

C. Amplitude Probability Density

Purpose: **To compute the amplitude probability density of sets of data in consecutive locations. An initialization subroutine and an execution subroutine are provided.**

Programmer and Date: **C. Cappello, July 1970**

Calling Sequence:

CALL BINIT, BLOCK (initialization routine)

CALL BMAIN, BLOCK (execution routine)

Input Parameters:

C(BLOCK+0) = pointer to input array

C(BLOCK+1) = number of points in input array

C(BLOCK+2) = upper bin limit

C(BLOCK+3) = ignored

C(BLOCK+4) = pointer to the bins

C(BLOCK+5) = lower bin limit

C(BLOCK+6) = bin width

C(BLOCK+7) = storage (BINIT fills with number of bins)

Registers Used: A, B, X, OF

Locations Used: 127_8, anywhere in core

Usage: Subroutine BINIT calculates the total number of bins from the formula:

$$N = \left\lceil 2 + \frac{\text{upper bin limit - lower bin limit}}{\text{bin width}} \right\rceil$$

and stores this in BLOCK+7. The symbols ⌈ ⌉ indicate that the rounded up integer value of the expression is to be used. In the above formula the 2 is there to accommodate numbers outside of bin limits. The user must leave enough room in core to accommodate all the bins. It is suggested that the user should plot the bin contents with the executive plotting subroutine for rapid examination.

Remarks:

After initialization, repeated calls may be made to the execution routine. The user should check OF on return to verify that no single bin overflowed. No indirect pointers are used.

D. FFT Spectrum Averaging

Purpose: To average a number of blocks of FFT output data in order to increase the accuracy of spectral analysis plots.

Programmer and Date: M. Saklad - February 1972

Input Parameters: C(C#DN) = N_{FFT} = number of data points per block

C(C#DN+2) = N_{BLOCK} = number of FFT output data blocks to be averaged where C#DN is accessed by @ DISP.

$N_{BLOCK(MAX)} = 32767_{10}$.

$N_{FFT(MAX)} = 4000_8 = 2048_{10}$.

N_{FFT} must be a power of 2.

Locations Used: 276_8 locations included in the FFT spectrum analysis program RESP. $6000_8 = 3072_{10} = 3*(N_{FFT(MAX)}/2)$ locations are used as a work area in lower core. These locations are not initialized unless RBLOCK averaging routine is to be called. This area is in addition to the $10000_8 = 4096_{10} = 2*N_{FFT(MAX)}$ locations normally utilized by RESP.

Usage: RBLOCK, which is incorporated as a subroutine of the FFT spectrum analysis program RESP, is executed only if $N_{BLOCK} > 0$. The resultant spectrum analysis will be displayed upon completion of the averaging. Time waveforms and phase plots are available only for the default condition of $N_{BLOCK} = 0$ in which the spectrum analysis is performed on only one block of data.

Example: In order to average $100_{10} = 144_8$ blocks of input data consisting of $1024_{10} = 2000_8$ data points, the following data must be provided:

@disp/xxxx 2000;	# of points per block (N_{FFT})
xxxx/xxxx;	input buffer address
xxxx/xxxx 144	# of blocks to be averaged (N_{BLOCK})
p/start	program start
c	

Error Messages: None. No error checking has been implemented for negative values of N_{BLOCK}.

Remarks: Upon execution, RBLOCK terminates the program table with the MULTICHIP display program; if the programs subsequent to MULTICHIP are desired, the simulation program must be reloaded since the RBLOCK work area utilizes the same core locations as these programs.

If it is desired to see time waveform plots and/or phase plots, N_{BLOCK} may be changed to zero and back again without problems. However, the data examined will be lost to the next averaging unless the change from $N_{BLOCK} = 0$ to $N_{BLOCK} = N$ is made immediately after the time waveform is plotted.

Program Operation: After a sufficiert number of input points (specified as the # of points per block) are available they are discrete Fourier transformed by the FFT routine, TRANS. The energy in each frequency band is obtained by calculating R^2+I^2 (real^2 plus imaginary^2 components). The energy in each frequency band of each block is added to the previous values. These values are calculated in double precision floating point to preserve accuracy and dynamic range. After this is done the specified number of times, the totals are normalized by the number of blocks in the average and converted to dB by taking 10 $\log_{10}$ (average) where the average is considered to be a B15 value. The equivalent bandwidth of each filter is given as (sampling rate/# of points per block); however, the program <u>does not</u> normalize by the filter bandwidth to give a true spectral density.

For a numerical example, assume the input data consists of independent samples of zero mean noise with standard deviation σ_o B0 (or $\sigma_{15} = \sigma_o * 2^{-15}$ B15 if the data is considered B15). Then the spectrum is flat with height, $H_c = (\#\text{ points per block}) * \sigma_{15}^2$. If enough blocks are averaged, the resultant spectral plot will be close to H_o. If $\sigma_o = 4096$ ($\sigma_{15} = 2^{-3}$), H_o will be (# of points per block)$*2^{-6}$; this number converted to dB, 10 $\log_{10}$ (# of points per block)$*2^{-6}$ is actually plotted. The standard deviation of the error for each spectral point will be approximately $\sigma_{H_o} \approx H_o * (2/\#\text{ blocks averaged})^{1/2}$. Hence averaging 100 blocks will yield $\frac{\sigma H_o}{H_o} \approx 0.14$ corresponding to 68% of the points falling within 0.6 dB (10 log 1.14) of the expected spectral level. This error analysis is true also for other than independent input samples.

E. Miscellaneous

The subroutines below are well enough commented in the listings that a brief description is adequate. For more complete information, reference should be made to the listings. In addition to corrections applied to the mathematics subroutines, error returns were deleted.

DP	Double-precision mathematics,
XSIN XCOS	Sine and cosine (modified to use table lookup),
XRCOS	Cosine with input angle in revolutions B15,
TARC	Phase angle of a complex number,
XSQRT	Square root,
AGSQRT	Pseudo-double-precision integer square root,
XLOG	Logarithm,
DBEX	Decibel conversion,
$OF	Routine which sets A register to saturation level if overflow occurred; this is frequently used after an addition,
#STR	Buffer transfer,
K7ADD	Block data adder,
K2CALL	Block data adder with scaling,
FUNC	Generalized function generator,
#GRAN	Random number generators,
#GGAUS	Gaussian random number generator,
#GP	Random bit generators,
TRAN	FFT generator.

V. SIMULATION PROGRAMS

A. DEBUG Commands for Use with the Simulation-Control Program

In order to simplify the use of the Simulation Control Program, a number of commands which refer symbolically to simulation parameters have been added to DEBUG. Communication between DEBUG and the Simulation Control Program is via the Item Table which is specified at assembly time. A parameter on the Item Table is described by a symbolic name (up to 4 characters) and includes information on the mode (e.g., octal, floating point), the minimum and maximum allowed values, and the CRT display routine for the parameter. The Item Table is loaded into memory along with the Simulation-Control Program; restarting DEBUG clears the Item Table.

In the descriptions below, ITEM represents the symbolic name of a parameter that has been assembled into the Item Table. All DEBUG operations with simulation parameters are prefixed with the symbol '@'. Parameter values are displayed in the mode specified in the Item Table entry for the parameter. When a carriage return is typed, DEBUG is restored to its former mode. The error message '??' will be typed in the following situations:

- after @, if there is no Item Table
- after the command, if ITEM is not on the Table
- after a new value is entered, if that value is outside the allowed range for the ITEM. In this case the original value is unchanged and is re-displayed by DEBUG.

<u>Command</u>: @ ITEM/

Action:

Display the first data value of ITEM in the proper mode. If ITEM comprises several data words, successive values may be examined or altered using the usual DEBUG commands (comma, semicolon, and colon).

Examples:[†]

@XXXX/100.0	Item XXXX has value 100.
I1000/LDA 100 c.r.l.f.	Examine location 1000 in instruction mode.
@XXXX/100.0 50 c.r.l.f.	Change XXXX to 50.
1000/LDA 100	Instruction mode is restored.
@YYZZ/??c.r.l.f.	Item YYZZ not found on Table.
@YY/1.234 ;	The first data value of YY is 1.234.
174/5.678 6.0;	(Note that trailing blanks in the item name need not be typed.)
175/9.012 10;??c.r.l.f. 175/9.012	The second data value is changed to 6.0 (This value happens to be stored in location 174.) An attempt is made to change the third data value of YY to 10, but this value exceeds the specified maximum.

Command: @ITEM$

Action:

Call the CRT display routine associated with ITEM. The routine will return to DEBUG. Error if no routine has been specified.

Command: @ITEM#

Action:

Record ITEM on magnetic tape, and return to DEBUG. (The tape drive will have been specified prior to execution of the Simulation Control Program.)

Command: @ITEM*

Action:

Enter ITEM on the Parameter-Stepping Table. Following the '*', DEBUG types '/' and displays the first of the three successive quantities: starting value, maximum value, step size. The user may enter or alter any of these quantities. If ITEM is

[†] In all examples, underlined characters are those typed by DEBUG. Carriage return and line feed, if not obvious, are denoted by c.r. and l.f. respectively.

already on the Stepping Table, DEBUG types the previously entered starting value; if ITEM is not on the Table, the starting value and the maximum value are set to 0.

There is space for 5 items on the Stepping Table.[†] An error message is typed if an attempt is made to enter a sixth item.

Example:

@XXXX÷/0.____1, 10, 1.6	**Item XXXX will be stepped through the values 1.0, 2.6, . . . , 9.0.**
@XXXX÷/1.0000___, 11	**The maximum stepping value for XXXX is changed to 11.0.**

Command: @÷

Action:

Clear the entire Parameter Stepping Table. (Individual items cannot be deleted from the table.) Restarting DEBUG clears both the Stepping Table and the Item Table.

[†] Items on the Stepping Table are handled by the Simulation-Control Program similar to nested DO loops in FORTRAN, with the first item corresponding to the innermost loop.

B. Simulation Tape Editor

Purpose: To copy selected information from Varian 620/i tapes written by the simulation program onto tapes in a format suitable for printing on the IBM 360.

Programmer: C. D. Cappello - January 1971

Input Parameters: Input parameter lists are requested by the following messages typed by the program. If there is more than one entry for a parameter list, the entries are separated by commas. A carriage return terminates each list; a carriage return with no entries indicates default values are to be used.

1. INPUT TAPE - tape drive (0-3) holding input tape. Drive must be specified before program can continue.
2. OUTPUT T, F - output tape and file number. 'S' indicates the listing is to be suppressed. 'N' indicates the next file from a previously specified tape and file number. Default value is next file if, on the previous run, an 'N' was typed or a tape and file number were typed; default value is suppress listing initially or if, on the previous run, an 'S' was typed.
3. DATA ITEMS a list of characters identifying items to be printed. Only the first 4 characters in each entry are recognized. Any number of items may be specified. Default value is all items.
4. TYPES - a list of type codes to be printed. Default value is all types.
5. PROGRAM NUMBERS - a list of program numbers to be printed. Default value is all numbers.
6. START AND STOP NOS - only information from runs between these numbers is printed. Default values are 1 and 77777_8.
7. FREQUENCY OF PRINTOUT - this number is used with reference to the first item in the data item list and permits the user to print a sampling of the tape at regular intervals. All occurrences of the specified data up to but not including the second appearance of the first item in the input item list are printed. All copying ceases until the first item appears k more times, where k is the number entered for this parameter. Thereafter, copying proceeds as usual until the next occurrence of the first item and ceases until the kth occurrence after that. As an example, suppose X, Y and Z are items that appear on the input tape in the following order:

   X, Y, Y, Z, Z, Z, X, Y, Y, Z, Z, Z, X, Y, Y, Z, Z, Z, ...

   If the frequency is specified as 3, and if X, Y, Z are inputs in the data item list, the underlined items in the above sequence are printed. Default value of this parameter is 1, which indicates that all occurrences of the specified data are copied.

Notes on Input Parameters:

A. An item is printed out only if it meets all requirements in 3-7.

B. 'Q' or 'R' may be entered in place of any numerical input and the program will return to DEBUG ('Q') or be restarted ('R').

Output:

An output tape is written for the IBM 360 consisting of one file for each set of input parameters. The output data may also be displayed on the CRT by enabling SS2. When a full page has been written the program waits for the user to t,pe the PAGE key before continuing. A 'Q' or an 'R' causes the program to terminate normally and then to quit or restart.

Format of the Printed Output:

Every run is printed starting on a new page. At the top of every page is printed the run number and the label information. Label information must be packed two ASCII characters per word and must be no more than 30 words. Items are printed with the item name on one line and the data on the following lines. The number of data words per line for an item is specified in the item table; is zero words per line are specified, the maximum number of words per line will be printed. If an error occurred in reading a particular item, an asterisk will appear before the item name, but the data, as read, will be printed.

Sense Switch Settings:

1. SS2 on - directs the data to the CRT as well as writing it on tape. Its setting may be changed at any time.
2. SS3 on - instructs the program to request all input from the teletype. It must be set prior to the execution of the program.

Usage:

This is a stand alone program requiring an input tape written by the simulation program. It first requests input parameters on the CRT or, if SS3 is on, on the TTY. It searches the input tape for specified data and copies this information onto the output tape in a format suitable for printing on the IBM 360. When either the specified number of files have been searched or the logical or physical end-of-tape is reached, the program rewinds the input tape and requests further input. Output may be viewed on the CRT when SS2 is on.

Error Messages:

1. ? an invalid character was typed. Re-enter the argument.
2. *SZ an invalid tape number was entered. Re-enter the argument.
3. RUN NO. TOO SMALL - the requested starting run number was smaller than the first run on the tape. The program is restarted.
4. RUN NOT ON TAPE - the requested starting run

number did not occur before the end of tape file. The program is restarted.

5. WRITE ERROR - either the output tape is offline or there is no write ring. Only the number of the output tape must be re-entered before the program continues.

Remarks: The simulation tape reading routines with no modification are assembled into this program.

Data Types: The following is a description of data types which must be specified for each item in the item table.

Type 0 - text. Data is interpreted as two ASCII characters per word. A maximum of 130 characters may be put on a line.

Type 1 - octal, maximum of 10 numbers per line.

Type 2 - fixed point. The position of the binary-point is specified in the code word. If the binary-point position is zero, output consists of full-precision numbers with a maximum of 14 on a line. Otherwise, output consists of numbers having only 4 significant figures with a maximum of 8 numbers per line.

Type 3 - binary, maximum of 7 numbers per line.

Type 4 - floating point, maximum of 8 numbers per line.

Type 5 - double precision fixed point. The position of the binary point is specified in the code word. A binary-point position of zero corresponds to the binary point between the two words. Output consists of numbers having 4 significant figures with a maximum of 8 numbers per line.

Type 6 - double precision integer. Output consists of full precision integers with a maximum of 10 numbers per line.

C. Simulation Tape Splicer

Purpose: To copy a number of files written in simulation system format from one magnetic tape to another. Copying begins at a specified run on the output tape and the copied run numbers are adjusted to follow in numerical order.

Programmer and Date: C. D. Cappello, June 1971

Input Parameters: Output tape number and run number on the output tape at which copying is to begin. Also the input tape number and the numbers of the runs on the input tape at which the copying is to start and stop.

Sense Switch Settings: SS3 - OFF for operation from the CRT.
ON for operation from the teletype.

Locations Used: 100_8 - 10236_8

Usage: In response to messages written on the CRT screen (or the teletype), the user enters the following octal parameters:

OUTPUT TAPE	(0-3)
OUTPUT STARTING RUN NO.	
INPUT TAPE	(0-3)
START AND STOP NOS.	

In response to the Output Starting Run No. message, the user must type a '1' if he wishes to copy onto a new tape. All files between the starting and stopping run numbers are copied. If no starting and stopping numbers are entered, but only a carriage return is typed, all files on the tape prior to the logical end of tape file are copied. If only a starting run number is entered, that one run is copied.

When copying of all specified files is completed, the program asks for a new input tape. Copying will continue in sequence on the output tape. In place of any input parameter, an 'R' may be typed and the program will be restarted.

When copying is completed, a 'Q' must be typed in place of one of the input parameters. At this time a logical end-of-tape file will be written on the output tape and the user will return to DEBUG.

Error Messages: ?

- an invalid character was typed (parameter must be re-entered)

*SZ

- the parameter entered exceeded allowable size

Run No. Too Small

- the specified output starting run number is less than the first run on the tape.

Run Not on Tape

- the specified output starting run number is greater than one larger than the last run on the tape.

Error Pos. Output Tape

- an error occurred in positioning the output tape at the specified run.

Fix Output Tape

- the output tape is offline or not write enabled. The program pauses for the tape to be fixed; a 'C' will continue the program.

Write Error

- an error occurred in writing on the output tape but the program continues.

End of Output Tape

- the physical end of tape has been reached. The program is restarted.

Parity

- a parity error has occurred in reading the input tape. Five tries are made before the message is typed. The program continues with the data as read.

Input Tape Offline

- the program pauses for the tape to be fixed. Typing a 'C' continues the program.

Read Error

- an error other than the above two occurred. Five attempts are made to read the record before the program continues with the data as read.

D. Miscellaneous

The features and routines below are well enough commented in the listing that a brief description is adequate. For more complete information, reference should be made to the listings.

ITEM TABLE

A table which establishes mnemonic identification of parameters and data to facilitate communication among the user, the Simulation program, and DEBUG.

PROGRAM TABLE

A table which determines the sequential order in which programs are to be executed.

RECORD TABLE

A table of item names which are recorded automatically on tape.

STEPPING TABLE

A table which allows the Control Program to vary parameter values as if they were placed into nested do-loops (see Section V,A).

SPECIAL STEP

A feature by which the Control Program can step parameters when 1) parameters are not identified in item table, 2) parameter values are not uniformly increased, or 3) multiple parameters are to be changed at one time.

TAPE READ
TAPE WRITE

Special tape-handling routines used by the Control Program. The read package is double buffered to minimize the time involved in data transfers, whereas the write package is single buffered, since maximum throughput rate is not needed for recording output data consisting of input parameters and performance statistics.

MULTICHIP

Subroutine to accumulate, store, and plot Fourier transforms of data which is generated in chip-length buffers.

DECO EDIT

A special routine which allows the user to edit out specific parts of a simulation output tape.